A Practical Application for Noise Power Spectrum Analysis

A Thesis

Submitted to the Faculty

of

Drexel University

by

Scott L. Cupp

in partial fulfillment of the

requirements for the degree

of

Master of Science in Biomedical Sciences

May 2008

Dedications

This thesis is dedicated to my wife, Erin. Your unconditional love and support throughout the years have made this possible.

I also dedicate this thesis to my children, Johanna and Rionna, the underlying force driving me to complete this thesis.

Finally, this is dedicated to St Joseph, whose invaluable intercession has helped make me who I am today

.

Acknowledgments

I owe an immense debt of gratitude to Andrew Maidment for all his time, patience, assistance and support with this project and my professional career. We have worked together for nearly a decade, and it continues to be an honor.

Thank you to Dr. Chang for agreeing to be part of this thesis project and for providing assistance through the process.

Finally, I would like to thank Mike O'Shea and Carla Garcia of the Hospital of the University of Pennsylvania Radiology Department, Clinical Physics section, for their constant moral support and for allowing me the opportunity to vent.

Table of Content

List of Tables

List of Figures

List of Text Boxes

Abstract

A Practical Application for Noise Power Spectrum Analysis

Scott L. Cupp

Advisor: Chang Chang PhD

Advisor: Andrew D. Maidment PhD

Noise power spectrum (NPS) analysis is a useful image quality metric. It provides a quantitative description of the amount and frequency of the noise fundamentally contained within a particular imaging system. The calculation of the NPS of an imaging system has been greatly facilitated recently with the use of modern mathematical programs, such as MATLAB®, which can quickly calculate the two dimensional Fast Fourier Transform (FFT) of an image. However an application that allows users to conveniently import image files, appropriately manipulate data and effortlessly generate one and two dimensional noise power spectra in a standardized way remains lacking. The goal of this project is to develop a graphical user interface (GUI) equipped application, utilizing MATLAB® as the underlying program, which will allow imaging scientists, as well as clinical diagnostic imaging medical physicists, to quickly and easily perform NPS analyses. The resulting application developed allows the user to import a variety of common image formats, load single or multiple image realizations, linearize image data using common methods, graphically or numerically select a region of interest (ROI), select subROI sampling area matrixes, calculate the NPS using two common zero-frequency reduction methods, and generate one-dimensional NPS graphs along the different frequency axes. Further development of this program will allow it to be used as a standalone application without the user requiring access to the full version of MATLAB®.

Chapter 1: Introduction

Background

Noise power spectrum (NPS) analysis is a useful image quality metric that provides a quantitative description of the amount and frequency of the noise fundamentally produced within a particular imaging system. With the recently increased use of digital systems within radiology departments, the need for a standardized and easily reproducible method for calculating NPS has come to the forefront. The calculation of the noise power spectrum of an imaging system has been facilitated by the use of modern mathematical programs, such as MATLAB (Version 7.4.0.287 "R2007a", Natick, MA), which can quickly calculate the two dimensional Fast Fourier Transform (FFT) of an image. However an application that allows users to conveniently import image files, appropriately manipulate data and effortlessly generate one and two dimensional noise power spectra in a standardized way remains lacking. The program developed for this project gives the clinical and research physicist, along with service engineers, a quality assurance tool which utilizes a simple graphical user interface (GUI) for calculating and visually interpreting the NPS of essentially any imaging system, which produces and stores a digital image and meets key mathematical requirements.

History

The noise characteristics of imaging systems have been studied over the course of many years with the modern era beginning with a 1949 publication by Sturm and Morgan[1] in which the statistical variance of optical density fluctuations within

radiographic images were characterized. During the 1950's, noise measurement theory and experimental methods were developed for photographic imaging systems by many authors including Jones,[2] Frieser,[3] and Zweig.[4] Computers equipment was not widely available, and specialized analog electronic noise power instrumentation had not yet been developed. An early technique was to mount a film in a frame, then move the film past an illuminated slit. The resulting fluctuations in transmitted light were then analyzed by an analog electronic spectrum analyzer (Doerner,[5] Doi,[6] Rossmann,[7] Lubberts,[8] and Vyborny *et al*.[9]). In later years, scanning digital microdensitometers became commercially available, but the cost and complexity of these microdensitometers limited their use to academic, government, and industrial laboratories.

The development of discrete NPS estimation techniques during the 1970's brought with it the need to consider the effects of data windowing. Windowing techniques are especially important in the case of narrow or closely spaced spectral peaks. There is the risk of blurring narrow peaks out into adjacent frequency bins. This is especially true when a narrow window is used, which has the effect of convolving a wide sinc^2 function with the NPS, therefore smoothing it. The use of windowing functions, including overlapped windowing techniques, has been extensively reviewed by Harris.[10]

As digital imaging evolved in the 1980s with the invention of storage phosphor-based digital radiography and other forms of digital radiography, the digital NPS estimation techniques had to evolve to include many special considerations such as: under-sampling the image and pixel size and shape distortion, which can result in aliasing

as described by Giger *et al*.[11] These effects can be quite significant under realistic operating conditions in digital radiography.[12]

NPS Calculation

For each position, (*x*,*y*), on a image, there is an associated signal value, $\tilde{a}(x,y)$. The tilde notation is used to denote that $\tilde{a}(x,y)$ is a random variable. Each acquired image or realization will differ slightly due to random fluctuations that are inherent within the image production chain. The NPS measures the fluctuations in the signal values in the image about the mean and the spatial correlation of these fluctuations. Assuming that the noise sources are wide-sense stationary (mean and autocorrelation are shift-invariant), the NPS or Wiener spectrum is defined as

$$W(u,v) = \lim_{\substack{X\to\infty \\ Y\to\infty}} \left\langle \frac{1}{2X}\frac{1}{2Y}\left| \int_{-X}^{+X}\int_{-Y}^{+Y} \Delta\tilde{a}(x,y)e^{-2\pi i(ux+vy)}dxdy \right|^2 \right\rangle \qquad (1)$$

where u and v are the spatial frequencies along the axes. Normally only finite regions can be used, resulting in a sample Wiener spectrum for a single realization found by

$$\tilde{W}(u,v) = \frac{1}{2X}\frac{1}{2Y}\left| \int_{-X}^{+X}\int_{-Y}^{+Y} \Delta\tilde{a}(x,y)e^{-2\pi i(ux+vy)}dxdy \right|^2 \qquad (2)$$

An estimate of the true NPS is found by averaging multiple realizations

$$\hat{W}(u,v) = \frac{1}{N}\sum_{i=1}^{N}\tilde{W}_i(u,v) \qquad (3)$$

where *N* is the number of sample Wiener spectra.

A discretely sampled imaging system produces an array of data, $\tilde{a}[m,n]$, where the integers m and n systematically index the sampling points. One can treat the NPS directly in terms of a "discrete time", "discrete space" or "digital" process[16,13,14]. This correspondence occurs because a sufficiently small sampling pitch will approximate the continuous process arbitrarily well. If the signal is bandwidth limited, then the continuous process can be recovered exactly. The continuous case and the discrete case are related by

$$\tilde{a}(x,y) = \sum_{m,n} \tilde{a}[m,n] g(x - mx_0) g(y - ny_0) \tag{4}$$

where x_0 and y_0 are the sampling pitches and g can be chosen as a Dirac delta function[15], a *sinc* function, or a *rect* function[16]. The discrete noise power spectrum (NPS) of a discrete random variable is defined as

$$W_d(u,v) = \lim_{\substack{N_x \to \infty \\ N_y \to \infty}} \left\langle \frac{x_0 y_0}{N_x N_y} \left| \sum_{m,n} \Delta\tilde{a}[m,n] e^{-2\pi i(umx_0 + vny_0)} \right|^2 \right\rangle \tag{5}$$

where N_x and N_y are the number of pixels along the axes in the region of interest. The sample NPS of a finite region for discrete frequencies u=k/$N_x$$x_0$ and v=l/$N_y$$y_0$ is found by

$$\tilde{W}_d[k,l] = \frac{x_0 y_0}{N_x N_y} \left| \sum_{m,n} \Delta\tilde{a}[m,n] e^{-2\pi i(km/N_x + ln/N_y)} \right|^2 \tag{6}$$

for $0 \le k \le N_x$ and $0 \le l \le N_y$. An estimate of the discrete NPS,

$$\hat{W}_d[k,l] = \frac{1}{N}\sum_{i=1}^{N}\widetilde{W}_{d,i}[k,l] \tag{7}$$

is obtained by averaging *N* sample Wiener spectra.

Assuming Gaussian statistics, the statistical error (a.k.a. spectral resolution) of the sampled NPS for both the continuous and discrete cases is

$$\sqrt{W_d[k,l]} \tag{8}$$

The statistical error, of a given frequency (*u,v*), for the continuous and discrete cases, for the estimated NPS is

$$\sqrt{\frac{W_d[k,l]}{N}} \tag{9}$$

where N is the number of sampled NPS. Thus, statistical accuracy can be increased by using a larger number of sample spectra to estimate the NPS.

Image Acquisition

Images to be used in the calculation of NPS should be of a uniform media consistent with the systems intended purpose and free of artifacts. Image acquisition should be performed in a standardized and repeatable manner. The specific details, which are at the discretion of the imaging scientist, will depend on the imaging modality and its intended purpose. A full exploration of image acquisition techniques is beyond the scope of this thesis, but a few general considerations should be taken during image acquisition. When calculating the NPS for quality assurance purposes the images should

be acquired using clinically relevant parameters for the particular imaging modality, such as: kVp, half value layer, pulse sequences and reconstruction algorithms. Typically images should be acquired over a range of signal intensities that corresponds to the range of clinically useful signal intensities. In addition, removal of the scatter control grids and the use of geometry that reduces backscatter should be implemented, if possible.

The NPS is commonly used in the calculation of the detector quantum efficiency (DQE)

$$DQE(f) = \frac{\Phi[k\ MTF(f)]^2}{NPS(f)} \tag{10}$$

where k is the gain factor of the system and Φ is the photon fluence. When acquiring images for this purpose, the technical parameters used should yield a known photon fluence.

Chapter 2: NPS Application

Loading Image(s)

The NPS application developed allows the user to utilize a GUI while MATLAB is utilized as the underlying program engine (Figure 1).

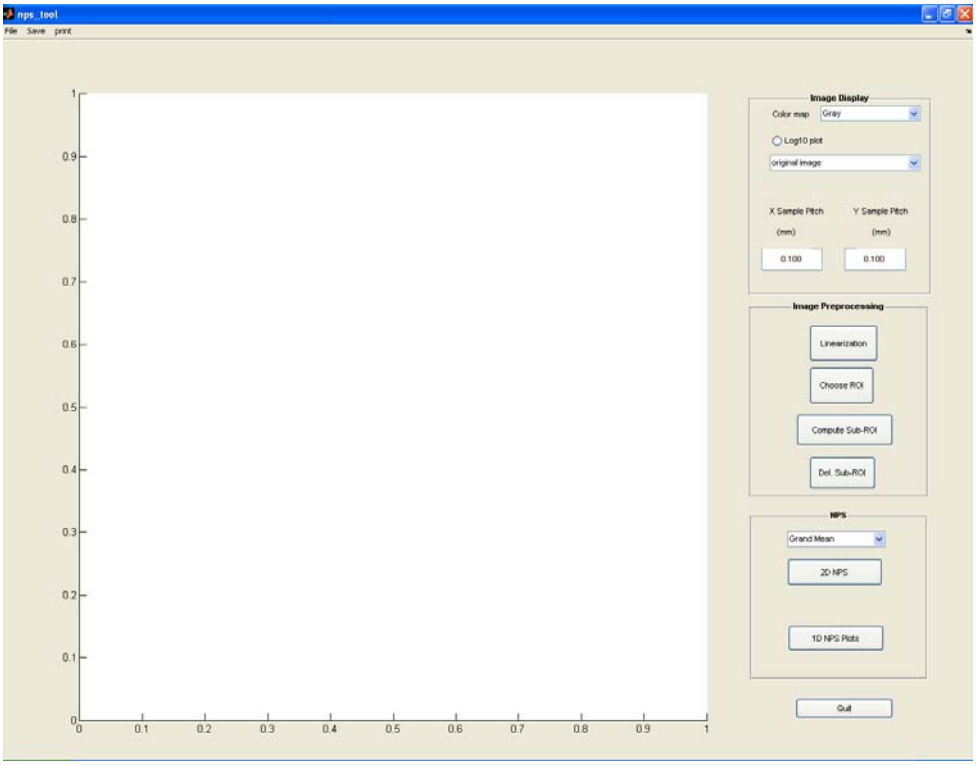

Figure 1: Graphical user interface of the NPS application

Images to be used with this application must be grayscale. There is no practical limitation to the image matrix size or to the bit depth used. The following common image file types may be used with this application: Graphics Interchange Format (*.gif); Joint Photographic Experts Group (*.jpg); Portable graymap (*.pgm); Sun Raster File (*.ras); Tagged Image File Format (*.tif); Dicom Image File Format (*.dcm). In addition

MATLAB binary files may be used. When an image file is selected, the application verifies that the selected image is grayscale. If a non-grayscale image is selected, an error message window will be produced and the image will not be loaded. When a grayscale image has been selected the image is loaded into memory and displayed. Images within the application maybe displayed using one of three color look up tables: Gray Scale, Hot or MATLAB default.

Linearization

The pixel values of the images acquired should be linear with the amount of signal captured. Some imaging systems commonly used in radiology perform a non-linear dynamic range compression on the image data to improve visualization of the image. The application provides the user two methods to re-linearize the image data (Figure 2). Additional methods can be trivially added.

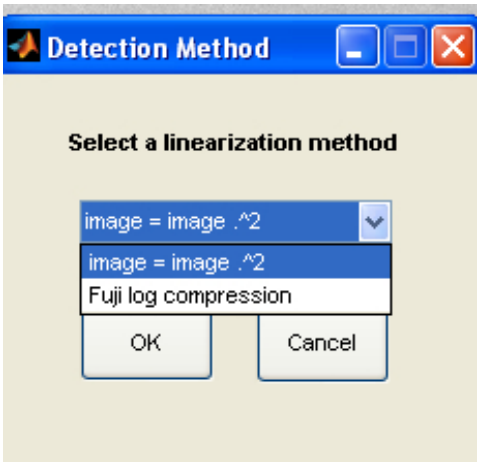

Figure 2: Displays the Graphical User Interface (GUI) for the selection of image data linearization.

Linearization must be completed before a ROI is selected and the application warns the user of this. The first option provided to the user is to square the image data, which will linearize square root compressed images, such as AGFA (Mortsel, Belgium) CR images. The second option provided to the user is to be used to correct Fuji Log Compression. Fuji (Toyama, Japan) imaging systems log compress the image data and shift the pixel value so that the center is at a value of 511. The image is re-linearized using

$$Sk = 4 - \log(\frac{Sensitivity}{4}) \tag{11}$$

$$Linear = 10^{(image - 511 + \frac{1024}{Latitude*Sk})(\frac{Latitude}{1023})} \tag{12}$$

where Sensitivity and Latitude are entered by the user. Sensitivity and Latitude are provided by the Fuji (or Fuji-based) imaging system for each image acquired. After linearization is complete the linearized image data is displayed for selection of a region of interest (ROI) and artifact analysis and.

ROI

There are many reasons to calculate the NPS only a limited portion of the acquired image. For example, some imaging systems may have permanent stationary artifacts or during the acquisition identifying marks may have been placed on the image. If any portion of these artifacts is included into the calculation of the NPS, the NPS will be incorrect. Therefore the program incorporates the ability to choice a ROI.

When the ROI function is launched the user is given the option to select the ROI by entering x-y coordinates or by graphically tracing out the ROI using a marquee tool.

X-Y coordinates are based on the matrix size of the image that has been loaded (Figure 3). When selecting the ROI the user should take care not to include any artifacts, if at all possible. If the ROI function is not utilized, the NPS will be calculated using the entire image matrix.

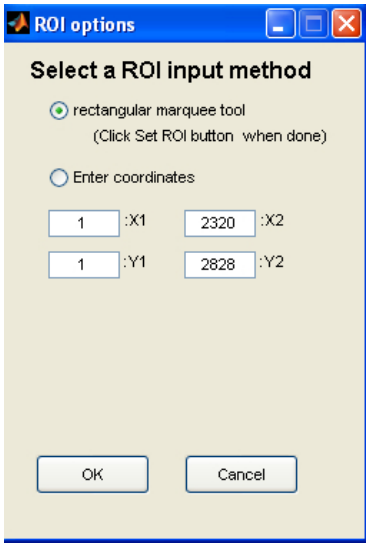

Figure 3: Displays the Graphical User Interface (GUI) for the selection of a Region of Interest (ROI).

Spectral & Frequency Resolution

Increasing the number of realizations will improve the statistical accuracy (spectral resolution) of the estimated NPS. Additional realizations can be obtained by acquiring multiple images from an ergodic system, or by subdividing a single image into smaller (subROI) regions if the system is wide sense stationary. The selection of a small subROI matrix will allow the creation of many NPS realizations from a single image and thus improving the spectral resolution by

$$1/\sqrt{N} \quad , \tag{13}$$

where N is the number of realizations, but at the cost of frequency resolution Decreasing the size of the subROI matrix used to calculate the NPS results in blurring of the estimated NPS, yielding a decreased frequency resolution. Frequency resolution is the ability to discern closely spaced frequency peaks. In an effort to reach the desired spectral resolution, one must often acquire and process multiple images, and subdivide each of those images in order to produces the hundreds of realizations needed to achieve adequate spectral accuracy. Ultimately it is the user's decision as to what is of the utmost importance to them; high spectral resolution, high frequency resolution or both with the use of a large number of large-pixel regions. The program gives the user the flexibility to do what is best for their specific situation.

To facilitate the production of multiple realizations, the program provides the user, via a dropdown menu, the option to select subROI sampling areas in matrixes of 8, 64, 128, 256 or 512 pixels squares (Figure 4).

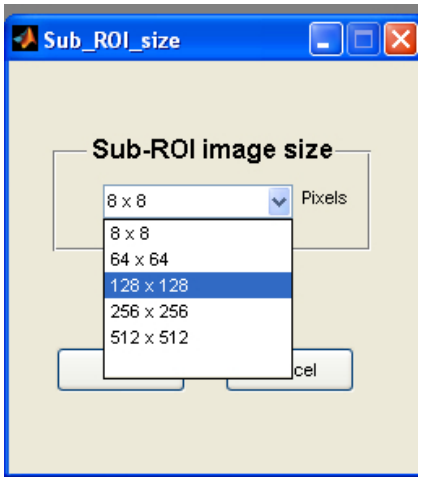

Figure 4: Displayed is the GUI for the selection of a subROI matrix. The selection of a smaller matrix will increase the number of realizations therefore improving the spectral resolution but at the cost of decreased frequency resolution.

Once selected, the application will subdivide the ROI using the specified matrix size, starting in the upper left corner, and will then graphically display the subdivided regions (Figure 5). The subROI matrix size selected must be smaller than the image or ROI matrix.

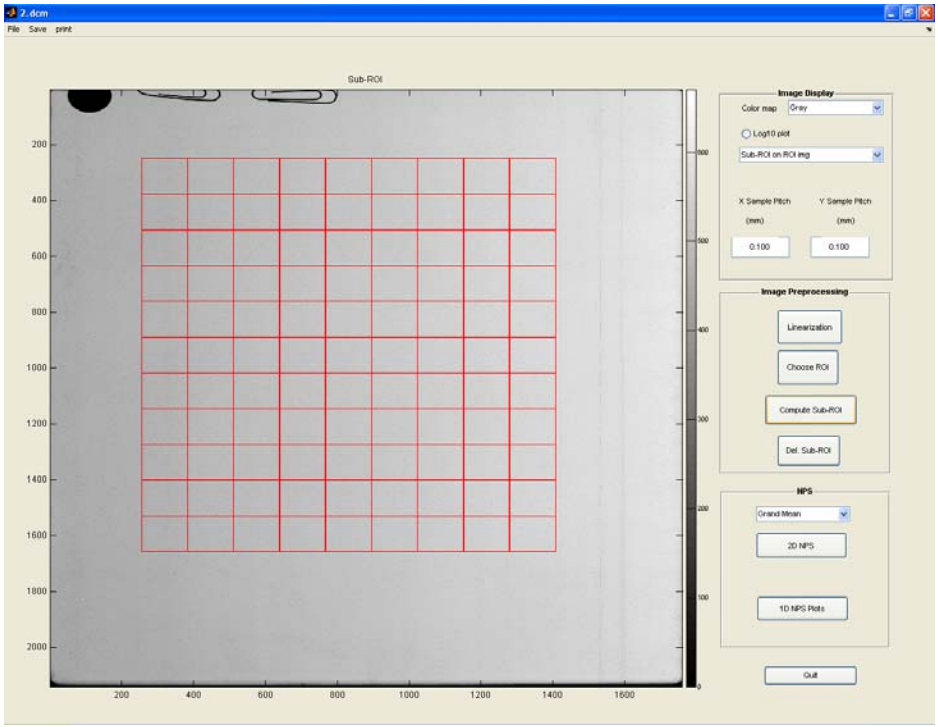

Figure 5: Displayed is a Computed Radiography (CR) image which has had a ROI selected and a 128x128 subROI matrix applied.

It is possible to have stationary artifacts, such as dead pixels, contained within the ROI. These artifacts should not be included in the NPS calculation. The program provides the ability to remove, or "delete," specific subROIs from the estimated NPS. The program graphically displays which subROI regions have been removed from the estimated NPS (Figure 6).

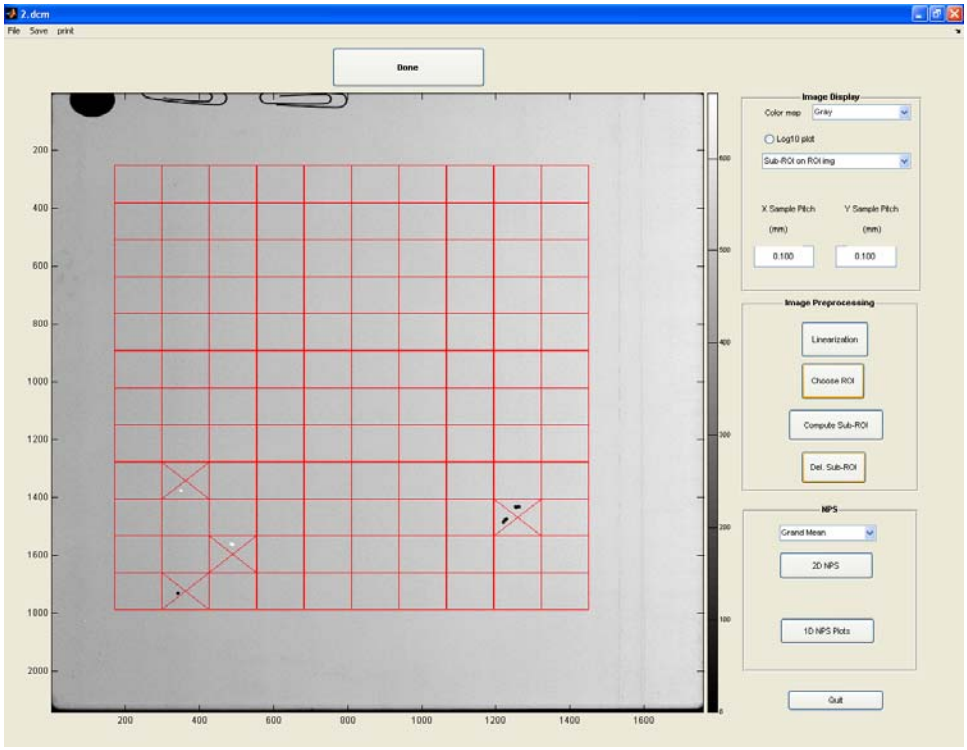

Figure 6: Displayed is a Computed Radiography (CR) image which has had a ROI selected and a 128x128 subROI matrix applied. The crossed out areas were subROI regions that were removed from the NPS averaging because of dust on the imaging plate which caused image artifacts.

Zero-Frequency NPS Estimation

Before the NPS can be calculated, the user must decide how to handle the zero-frequency or DC component estimation. The DC value of the NPS is the average signal intensity in the image. By its very nature, this value will fluctuate from image to image based on the stochastic generation of the image data. However, the DC component is often contaminated with non-stochastic effects, such as variations introduced by the x-ray generator or the image detector. Such effects can introduce a zero-frequency spike in the NPS that does not truly reflect the stochastic noise. In order to avoid a biased estimate of

the zero-frequency component of the NPS, user control of the mean pixel value is provided before the NPS is calculated. The user has three options, provided via a dropdown menu, for the estimation of the zero-frequency component (Figure 7).

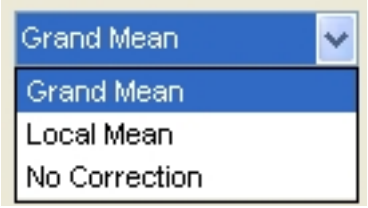

Figure 7: Displayed is the GUI for the selection of which zero-frequency component estimation method is to be used during the calculation of the NPS. The Grand Mean method subtracts the mean of the entire ROI. The Local Mean method subtracts the mean of the individual subROI regions from itself.

The first option provided, ”No Correction,” allows the user the process the NPS without alteration of the mean pixel values from the image. “No Correction” is useful in the most ideal of situations, such as simulated image data. The second option, “Grand Mean” method subtracts the mean of the full ROI from the image before calculation of the NPS. If an ROI was not selected, the mean of the entire image is used. This method is sufficient when the average intensity of the background is uniform over the entire image or ROI; that is, the image is not subject to non-stochastic effects, such as the anode heel effect. The grand mean is calculated using the ROI image, hence removing any artifacts from the mean calculation (Text Box:1). The grand mean method should only be applied when multiple subROIs are selected for each ROI. In general, a large number of subROIs are required to obtain an unbiased estimate of the noise.

Text Box 1: NPS application MATLAB code fragment showing the calculation of the "Grand Mean" method for estimating the magnitude of the zero-frequency components. handles.sub_rois_axis refers to the boundaries of the ROI region.

```
if handles.mean == 0  % Calculates "grand mean"
c=handles.sub_rois_axis;
roi_mean=mean(mean(img(c(3):c(4),c(1):c(2))));
img(c(3):c(4),c(1):c(2))=img(c(3):c(4),c(1):c(2))-roi_mean;
```

In many instances the image may be affected by deterministic effects resulting in the average background intensity being non-uniform over the entire image or the ROI. Examples of common deterministic effects causing non-uniform background can include the anode heel effect, non uniform detector sensitivity or geometric angulations of the x-ray source and/or detector array. When the average background intensities trend slowly over the image, subtraction of the grand mean from the image will still result in an inaccurate estimation of the zero-frequency component of the NPS. The Local Mean method subtracts the mean of the individual subROI sampling areas from themselves (Text Box: 2), effectively setting the zero-frequency component to zero. If subROIs are not used, then the Grand mean and the local mean methods are equivalent.

Text Box 2: NPS application MATLAB code fragment showing the calculation of the "Local Mean" method for reducing the magnitude of the low frequency components. sub_rois{i} refers to the boundaries of the i subROI region.

```
for i=1:size(sub_rois,2)
 if ~sub_rois_del(i)
  c=sub_rois{i};
  roi_mean=mean(mean(img(c(3):c(4),c(1):c(2))));
  img(c(3):c(4),c(1):c(2))=img(c(3):c(4),c(1):c(2))-roi_mean;
```

NPS Calculation

The calculation of NPS, as explained in equation 6, is the absolute value of the square of the two-dimensional Fast Fourier Transform (FFT) of the image, which is then scaled by the product of the sampling pitch divided by the size of the matrix. The result produced by MATLAB function "fft2" is such that the zero-frequency is at the origin and mirrored along the horizontal and vertical axes. The MATLAB function "fftshift" must therefore be utilized to rearrange the output of the "fft2" so that the zero-frequency components are centered in the image (Text Box 3) with the higher frequency components along the periphery.

Text Box 3: NPS application MATLAB code fragment showing the calculation the estimated NPS using subROI regions and the "Grand Mean" method.

```
      for i=1:size(sub_rois,2)
         if ~sub_rois_del(i)
            c=sub_rois{i};

fft2d=fft2d+(abs(power(fftshift(fft2(double(img(c(3):c(4),c(1):c
(2))))),2)))*((x_del*y_del)/Size);
            count=count+1;

         end
      end
      fft2d=fft2d./count;
```

This is useful for visualizing symmetry of the two dimensional NPS. Once the NPS has been calculated, the results are displayed in a two-demensional image (Figure 8).

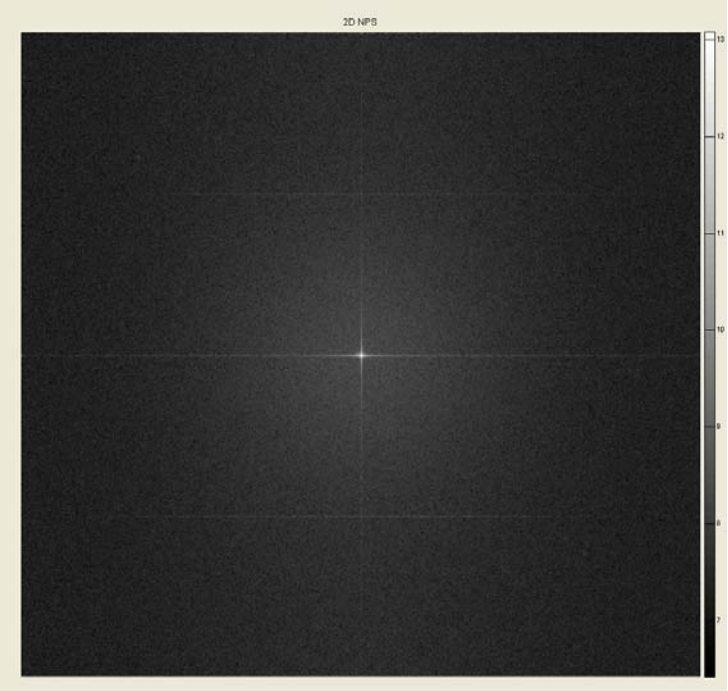

Figure 8: Two dimensional NPS with the zero-frequency centered and the highest frequencies along the periphery.

One Dimensional NPS Plots

In addition to displaying the NPS two dimensionally, one-dimensional line graphs are traditionally used in the literature. The NPS along the x and y frequencies are graphed, as well as the NPS along the 45-dregree between the x and y frequencies (Figure 9).

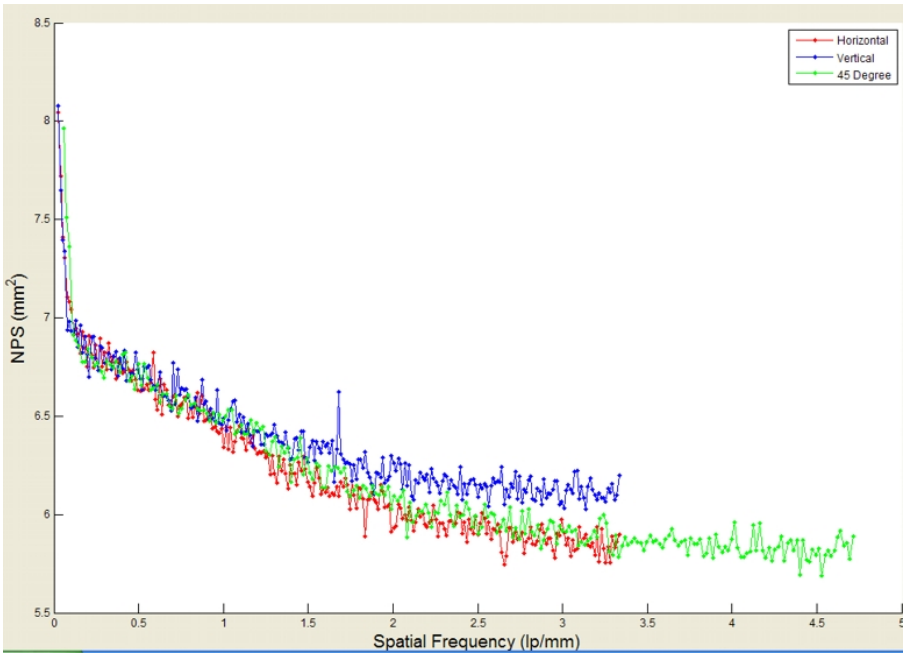

Figure 9: Default one-dimensional NPS line graph with the vertical, horizontal and 45-degree frequencies displayed.

The frequency along the 45-degrees is simply equal to

$$\sqrt{\left(u^2+v^2\right)}, \tag{10}$$

where *u* and *v* are the frequencies in the x and y directions respectively. The NPS along the 45-degree is used since the noise can be resolved to a higher frequency. The trend of the 45-degree frequency can indicate if there is aliasing present. As mentioned earlier, the fact that finite regions are used to calculate the NPS, produces a truncation artifact when the NPS contains frequencies above the sampling frequency. If there are noise frequencies above the sampling frequency, they are misrepresented as lower frequencies. If the 45-degree NPS frequency trends higher past the vertical and horizontal frequencies, this indicates the possibility of aliasing being present.

There are often additional noise components along the abscissa and the ordinate, which are associated with image readout or processing and hence non-stochastic. When the one-dimensional NPS line graphs are generated it is typical to exclude the frequencies along the abscissa and the ordinate. Rather, the average of the lines above and below the axis of interest are calculated and used for creation of the NPS line graphs. The program allows the user to select the number of lines above and below the axis of interest. Increasing the number of lines used in the averaging will have the effect of smoothing out the NPS line graphs, but also reduces the spectral accuracy (Figure 10).

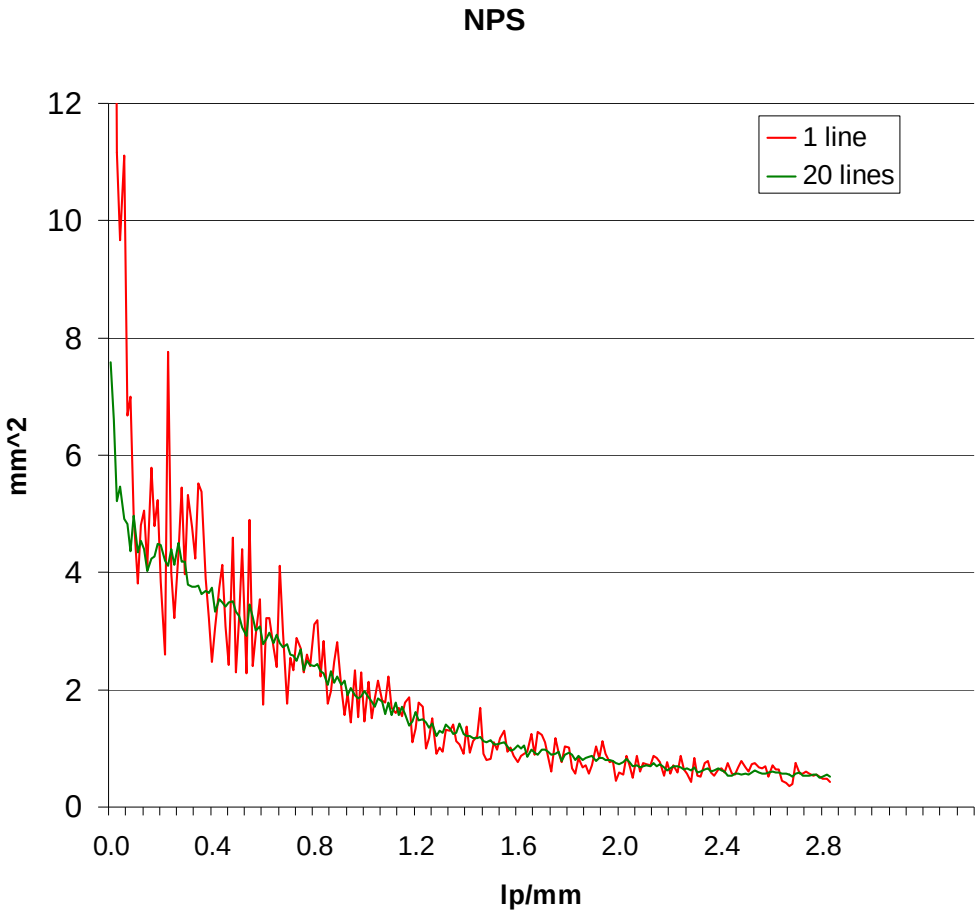

Figure 10: One dimensional NPS illustrating the smoothing effect caused by increasing the number of lines above and below the axis which are averaged together.

As mentioned previously, it is often necessary to acquire multiple images to provide adequate sample NPS to increase the spectral resolution of the estimated NPS. The program allows the user to load multiple images by utilizing the "Open Batch Images" under the file menu. There is no limit to the number of images to be loaded for calculation of the NPS estimation. The user must use caution while loading a large number of images, which are typically multiple megabytes each, because of the drain on

computer resources. Loading of fifty 3.5 MB images on a 3GHz Pentium 4 with 1 GB of memory took approximately 5 minutes with all other applications closed.

The application provides the user with a variety of options for saving the data generated. The ROI may be saved as either a .mat or .tif file. The one and two dimensional NPS images may also be outputted to a .mat file. The user also has the ability to print the current displayed graphic to a printer or to a .jpg file.

Chapter 3: Validation

In order to determine if the developed program was calculating NPS properly while different program features were utilized, the following procedures were performed. The program's output was then compared to the expected results.

Constant background no noise

The simplest scenario of constant background intensity without noise added was checked to verify program output. A 2048x2048 MATLAB image matrix was created with a constant element value of 2000. The NPS was calculated with the x and y sampling pitch set to 1.0 mm. It was calculated without a ROI or subROI regions being selected, then again with a ROI and a range of subROI sampling regions selected. The later was repeated for both the "Grand" and "Local" Mean methods. All of the NPS calculated had the excepted result of a uniform value of zero. The matrix was then processed again, but this time using the "No Correction". The calculated NPS was the expected delta function at the zero-frequency and zero elsewhere.

Constant background with white noise added

As stated earlier the error associated with the estimated NPS decreases as the number of realizations increases. To verify that the program performs as expected, a 2048x2048 MATLAB white noise image matrix was created with a known mean of 2000 and a $\sigma = 40$ which produced a realization with a mean of 2018.5 and $\sigma = 39.73$. The NPS of the image was calculated with the x and y sampling pitch set to 1.0 mm and

without an ROI. The NPS was calculated without using any subROI regions resulting in a single realization. The NPS was then calculated using subROI regions of 512, 256, 128 and 64 square pixels which resulted in 16, 64, 256 and 1024 realizations respectively. The standard deviation of each of the estimated NPS was calculated. The standard deviation of the single realization was scaled by $1/\sqrt{N}$ and then compared to the measured standard deviation. The measured standard deviation compared to the predicated by no than 0.58% (Table 1, Figure 11).

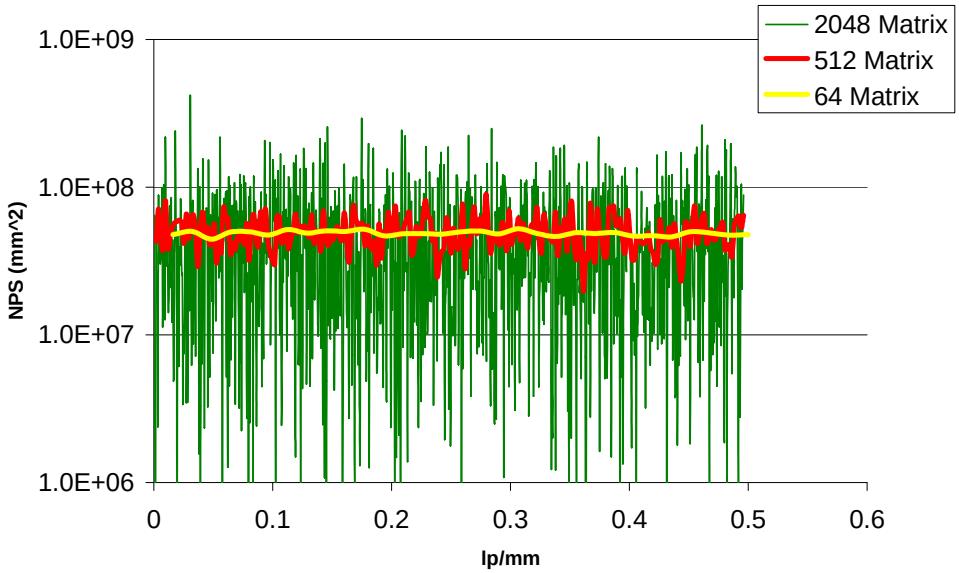

Figure 11: Demonstrating the effect of decreasing the subROI size. The 64x64 subROI region demonstrates significantly lower noise compared the 2048x2048 matrix.

Table 1: A comparison of predicted standard deviation to the measured standard deviation of the estimated NPS. As the number of realizations increases the standard deviation decreases as predicted.

Matrix	Number	(1/N)^0.5	Calculated StdDev	Measured DtdDev	% Error
2048	1	1	4.8618E+07	4.8618E+07	
512	16	0.25	1.2155E+07	1.2139E+07	0.13%
256	64	0.125	6.0773E+06	6.0910E+06	-0.23%
128	256	0.0625	3.0386E+06	3.0211E+06	0.58%
64	1024	0.03125	1.5193E+06	1.5139E+06	0.36%

Grand Mean vs. Local Mean Method

As mentioned earlier, when an image has a deterministic effect which produces a slowly changing average background intensity, the local mean is the preferred method. To verify that the program produces this expected result, the NPS of an image from a computed radiography system demonstrating slowly changing average background intensities, caused by the anode heel effect, was calculated. A ROI was selected to exclude the periphery and a 128 subROI matrix was selected. The NPS was calculated using both the Grand Mean method and the Local Mean method. The result of the Local Mean method was subtracted from the Grand Mean method. The zero-frequency was the only frequency that changed, as expected (Table 2). Note that only the frequency bins nearest the zero frequency are shown; however, the observed effect applied to all non-zero frequency bins.

Table 2: A comparison of the Grand Mean versus the Local Mean methods displaying the center portion the spectra. The Grand Mean and Local Mean methods produce identical results except at zero-frequency, where the local mean method equals zero. This is the expected result.

lp/mm	-0.104	-0.052	0.000	0.052	0.104
Grand Method NPS	1.378E+09	6.886E+09	1.220E+12	6.886E+09	1.378E+09
Local Method NPS	1.378E+09	6.886E+09	0.000	6.886E+09	1.378E+09
Grand-Local Method	0.000	0.000	1.220E+12	0.000	0.000

Known Noise Frequency

To test the frequency accuracy of the NPS program a 1024x1024 MATLAB file with 0.5 and 3.0 lp/mm noise sources was created. The estimated NPS was calculated with 128x128 subROI regions, producing 64 realizations and utilizing the Grand Mean method. The results showed the noise frequencies at the appropriate frequencies (Figure 12).

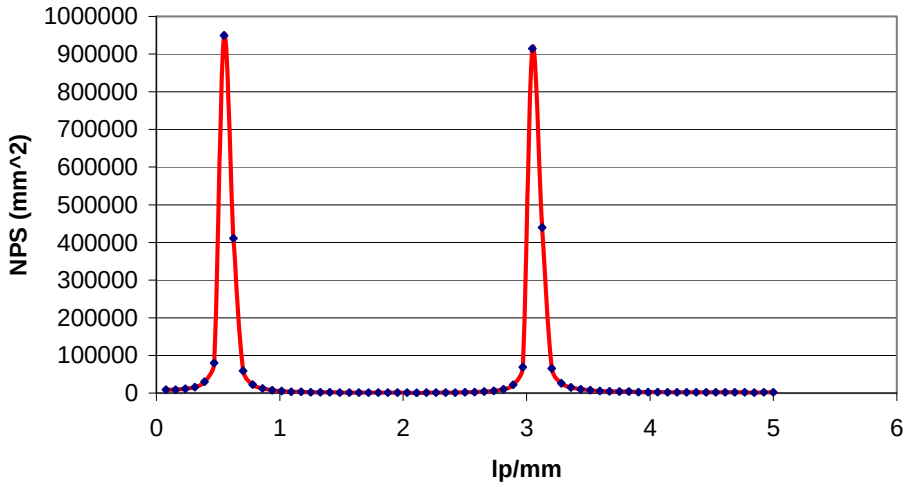

Figure 12: An image with a known noise source of 0.5 and 3.0 lp/mm was created and analyzed to verify the accuracy of the program. The NPS was calculated using 128x128 subROI matrix and the Grand Mean method.

Known 8x8 Calculation

In the American Association of Medical Physicist Task Group 16 Draft report "Standard for Measurement of Noise Power Spectra" dated February 11, 2004[17]; a simple 8x8 image matrix was provided for which the NPS has been calculated and is considered to be a known solution. An image file was created using the given 8x8 data set. As per the draft report the known mean of 2000, which is different from the true mean, was subtracted from the image before calculating the NPS. The subtracted image file was then processed through the program to calculate the NPS without using a ROI or subROI sampling region and setting the x and y sampling pitch to 1.0. The resulting NPS was exported for comparison to the known result. The calculated results correspond with the known results. (Table 3)

Table 3: In order to confirm the precision of the NPS program developed its output was compared to a known result. (A) Simulated 8x8 image matrix. (B) Calcualted NPS that is considered a "known" solution. (C) The NPS calculated by the NPS programs developed. The output of the NPS program compares to that of the "known" results. The highlighted cell different from the given by 0.01, which is attributed to a rounding difference.

A. Sample image data provided by AAPM TG16

2002	2017	1970	1997	1971	2017	2015	1983
1985	1976	2005	1985	2024	2034	1963	1979
2079	2009	1963	1998	1960	1986	1999	2034
2000	1963	1969	2018	1992	1958	1999	2036
1984	1995	1976	1990	2017	2019	2008	1994
1999	1981	1974	2009	1966	2000	2006	2008
1978	1972	2000	1954	1979	1984	1953	2023
2003	1946	1994	2008	2030	1995	2001	2030

B. Calculated NPS provided by AAPM TG16

162.56	86.69	945.62	1066.74	1.56	1066.74	945.62	86.69
255.52	928.35	417.04	535.75	899.73	1472.58	18.10	1523.44
116.50	724.87	1735.81	1338.98	146.25	449.45	915.31	128.59
28.54	94.87	690.21	366.90	337.58	80.94	21.77	704.67
280.56	274.74	1061.12	1327.45	1785.06	1327.45	1061.12	274.74
28.54	704.67	21.77	80.94	337.58	366.90	690.21	94.87
116.50	128.59	915.31	449.45	146.25	1338.98	1735.81	724.87
255.52	1523.44	18.10	1472.58	899.73	535.75	417.04	928.35

C. NPS calculated by the program.

162.56	86.69	945.63	1066.74	1.56	1066.74	945.63	86.69
255.52	928.35	417.04	535.75	899.73	1472.58	18.10	1523.44
116.50	724.87	1735.81	1338.98	146.25	449.45	915.31	128.59
28.54	94.87	690.21	366.90	337.58	80.94	21.77	704.67
280.56	274.74	1061.13	1327.45	1785.06	1327.45	1061.13	274.74
28.54	704.67	21.77	80.94	337.58	366.90	690.21	94.87
116.50	128.59	915.31	449.45	146.25	1338.98	1735.81	724.87
255.52	1523.44	18.10	1472.58	899.73	535.75	417.04	928.35

Chapter 4: Application of Program

The NPS program developed can be a useful tool for both the clinical and research settings. One such example of its use as a research tool is in the testing of a new research digital tomosynthesis system. During the initial setup of the system the research imaging physicist suspected the presence of some underlying noise in reconstructed phantom images. They acquired a set of flat field images which were then reconstructed into 56 slices. Each of the 56 slices should only contain a random variation in recorded intensity do to the x-ray production. The 56 slices were loaded into the NPS application using the batch operation. A ROI was selected in the center of the images avoiding the periphery were there are known reconstruction artifacts. A 128x128 subROI matrix was selected and the NPS was calculated using the grand mean method. The resulting two dimensional (Figure 13) and one dimensional (Figure 14) NPS showed periodic noise along the vertical frequencies with a fundamental frequency of approximately 0.5 line-pair per millimeter.

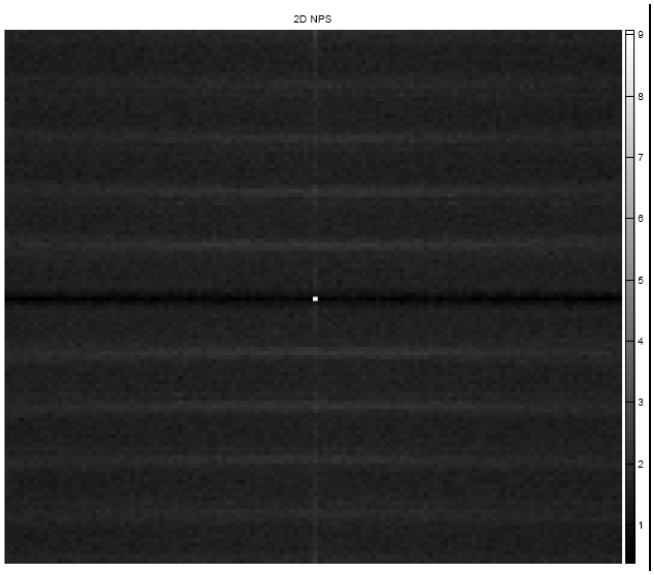

Figure 13: Two dimensional NPS average of 56 reconstructed flat field slices from a research digital tomosynthesis system. The NPS was calculated using a ROI of the center of the images and a 128x128 subROI matrix. The periodic vertical banding indicated that the system has some unknown intrinsic noise added to the reconstructed images along the vertical spatial domain.

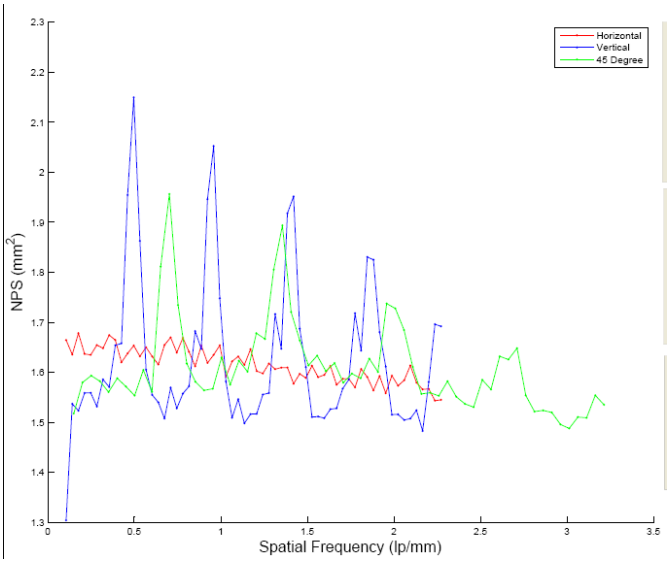

Figure 14: One dimensional NPS average of 56 reconstructed flat field slices from a research digital tomosynthesis system. The 1D NPS was calculated using a ROI of the center of the images, a 128x128 subROI matrix and averaging ±20 from the origins. The periodic vertical banding with a fundamental frequency of 0.5 line-pair per millimeter indicated that the system has some unknown intrinsic noise added to the reconstructed images along the vertical spatial domain.

Chapter 5: Further Development and Conclusion

Currently the application is limited to displaying one image within the set of multiple or "batch" images to be processed. This one image may be used to select the ROI, subROI sampling area matrix and to delete specific subROI sampling areas from the NPS calculation. This is sufficient as long as the acquisition geometry remains constant and the imaging system is reproducible. This will become problematic when each realization requires different subROI deletions. Such subROI deletions may be required if there are randomly occurring artifacts such as random pixel drop-outs. Future improvements will give the user the option to scroll through each image to delete different subROI regions from the NPS estimation.

The current version of the application only has two options for linearizing image data. Additional linearization options should be included in futures versions, such as the ability for the user to define simple linearization equations without having to modify the code.

Currently the application processes two options for estimating the zero-frequency noise components. It has been suggested by Flynn *et al* that a low order polynomial correction will also help with the removal lower frequency components. A polynomial fitting function will be added in subsequent versions[18].

The NPS application developed is a useful tool for the analysis of noise power spectrum in the clinical and research settings. Future versions of the application will enhance its ability to assist the imaging scientist. It is the ultimate goal of this project to produces a stand alone executable version of the NPS application that does not require

the user to have a licensed version of MATLAB. Thus, allowing all clinical medical physicists and imaging scientist to freely use the application for noise power spectrum analysis.

List of References

[1] R. E. Sturm and R. H. Morgan, "Screen intensification systems and their limitations," Am. J. Roentgeneol., vol. 62, pp. 617-634 (1949).

[2] R. C. Jones, "New method of describing and Measuring the Granularity of Photographic Materials," J. Opt. Soc. Am., vol. 45, pp. 799-808, (1955).

[3] H. Frieser, "Noise Spectrum of Developed Photographic Layers Exposed by Light, X-Rays and Electrons," Photogr. Sci. Eng., Vol. 3, pp 164ff, (1959)

[4] H. J. Zweig, "Autocorrelation and Granularity. Part I. Theory," J. Opt. Soc. Am., vol. 46, pp 805ff (1956); H. J. Zweig, "Autocorrelation and Granularity. Part II. Results on Flashed Black and White Emulsions," J. Opt. Soc. Am., vol. 46, pp 812ff (1956); H. J. Zweig, "Autocorrelation and Granularity. III. Spatial Frequency Response of the scanning System and Granularity Correlation Effects Beyond the Aperture," J. Opt. Soc. Am., vol. 49, pp 238ff (1959).

[5] E. C. Doerner, "Wiener Spectrum Analysis of Photographic Granularity," J. Opt. Soc. Am., vol. 52, pp. 669-672 (1962).

[6] K. Doi, "Wiener Spectrum Analysis of Quantum Statistical Fluctuations and Other Noise Sources in Radiography" in *Television in Diagnostic Radiology*, R. D. Moseley and J. H. Rust, eds., (Aesculapius, Birmingham, AL, 1969), pp. 313-333.

[7] K. Rossmann, "Modulation Transfer Function of Radiographic Systems Using Fluorescent Screens," J. Opt. Soc. Am., vol. 52, pp. 774-777 (1962). K. Rossmann, "Measurement of the Modulation Transfer Function of Radiographic Systems Containing Fluorescent Screens," Phys. Med. Biol., vol. 9, pp. 551-557, (1964).

[8] G. Lubberts, "Random Noise Produced by X-Ray Fluorescent Screens," J. Opt. Soc. Am., vol. 58, pp. 1475-1483 (1968).

[9] C. Vyborny, L.-N. Loo and K. Doi, "The energy dependent behavior of noise Wiener spectra in their low-frequency limits: Comparison with simple theory," Radiology, vol. 144, pp. 619ff.

[10] F. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform", Proc. IEEE, vol. 66, pp. 51-83 (1978).

[11] M. Giger, K. Doi, and C. Metz, "Investigation of Basic Imaging Properities in Digital Radiography: 1. Noise Wiener Spectrum," Med. Phys., vol. 11, pp. 797-805 (1984).

[12] M. Giger and K. Doi, "Analysis of MTFs, Wiener Spectra, and Signal-to-Noise Ratios of Digital Radiographic Imaging Systems," in Recent Developments in Digital Imaging, (K. Doi, L. Lanzl, P.-J. Lin, eds), pp. 60-81, (AAPM; American Institute of Physics, New York) , (1985).

[13] Oppenheim, A.V., R.W. Schafer, and J.R. Buck, Discrete Time Signal Processing, 2nd ed. (Prentice Hall, Upper Saddle River, NJ, 1999).

[14] M. Albert, and A.D.A. Maidment. Linear Response Theory for Detectors Consisting of Discrete Arrays. Medical Physics, **27**(10), 2417-2434, October 2000.

[15] Cunningham, I.A., "Applied Linear-Systems Theory", in Handbook of Medical Imaging, Vol. 1., edited by J. Beutel, H.L. Kundel, and R. Van Metter, SPIE Press, Bellingham WA, 2000, pp.79-159

[16] Papouis, A. Probability, Random Variables, and Stochastic Processes, 3rd ed., McGraw-Hill Inc (1991). pp. 332-356, 373-376.

[17] Maidment *et al* "Standard for Measurement of Noise Power Spectra" AAPM Report: Report of Diagnostic Imaging Task Group No. 16 Draft version 3.0 (2004)

[18] M. Flynn and E. Samei, "Experimental Comparison of Noise and Resolution for 2k and 4k Storage Phosphor Radiography Systems," Med. Phys., vol. 26, no. 8, pp. 1612-1623 (1999).

Appendix A: MATLAB Code for NPS Application

```
function varargout =nps_tool(varargin)
%NPS_TOOL M-file for nps_tool.fig
%      NPS_TOOL, by itself, creates a new NPS_TOOL or raises the existing
%      singleton*.
%
%      H = NPS_TOOL returns the handle to a new NPS_TOOL or the handle to
%      the existing singleton*.
%
%      NPS_TOOL('Property','Value',...) creates a new NPS_TOOL using the
%      given property value pairs. Unrecognized properties are passed via
%      varargin to nps_tool_OpeningFcn.  This calling syntax produces a
%      warning when there is an existing singleton*.
%
%      NPS_TOOL('CALLBACK') and NPS_TOOL('CALLBACK',hObject,...) call the
%      local function named CALLBACK in NPS_TOOL.M with the given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help nps_tool

% Last Modified by GUIDE v2.5 19-May-2008 16:32:51

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @nps_tool_OpeningFcn, ...
                   'gui_OutputFcn',  @nps_tool_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
  gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before nps_tool is made visible.
function nps_tool_OpeningFcn(hObject, eventdata, handles, varargin)
  % This function has no output args, see OutputFcn.
  % hObject    handle to figure
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)
  % varargin   unrecognized PropertyName/PropertyValue pairs from the
  %            command line (see VARARGIN)

  % Choose default command line output for nps_tool
  handles.output = hObject;

  movegui(handles.nps_tool,'north');
  handles.axes1_pos=get(handles.axes1,'Position');
  set(handles.set_roi,'visible','off')
  %shows status of the module
  handles.roi_state=0;
  handles.sub_roi_rm=1;
  %save the starting handles info
  tmp=handles;
  handles.start_condition=tmp;

  % Update handles structure
  guidata(hObject, handles);

  % UIWAIT makes nps_tool wait for user response (see UIRESUME)
  % uiwait(handles.nps_tool);


% --- Outputs from this function are returned to the command line.
function varargout = nps_tool_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in quit.
function quit_Callback(hObject, eventdata, handles)
% hObject    handle to quit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
   selection = questdlg(['Quit nps_tool ?'],...
      ['Quit mtf_tool...'],...
      'Yes','No','Yes');
   if strcmp(selection,'No')
      return;
   end

   delete(handles.nps_tool)
```

```
% --- Executes on selection change in plot.
function plot_Callback(hObject, eventdata, handles)

 if ~isfield(handles,'img') %|| ~(handles.roi_state) %|| ~isfield(handles,'img')
    return
end

axes(handles.axes1);
cla reset; %clear the axes for plotting

%reset axes1 size
figpos=handles.axes1_pos;
%position = [left bottom width height];
set(handles.axes1, ...
  'Position',[figpos(1) figpos(2) ...
  figpos(3) figpos(4)],...
  'FontSize',9);


contents = get(hObject,'String');
switch contents{get(hObject,'Value')}
  case ('original image')
    plot_bar(handles,handles.img);
    title(handles.filename,'Interpreter','none')
  case ('ROI image')
    %plot_bar(handles,handles.img_roi);
    plot_bar(handles,handles.img);
    axis(handles.img_roi_axis)
    title('Rescaled ROI')
  case ('Sub-ROI on ROI img')
    if ~isfield(handles,'sub_rois')
      h=msgbox({'Sub-ROI not selected yet!'},...
        'Instruction','help','modal');
      waitfor(h)
      return
    end
    %plot_bar(handles,handles.img_roi);
    plot_bar(handles,handles.img);
    sub_rois=handles.sub_rois;
    sub_rois_del=handles.sub_rois_del;
    hold on
    for i=1:size(sub_rois,2)
      tmp=sub_rois{i};
      x=[tmp(1) tmp(2) tmp(2) tmp(1) tmp(1)];
      y=[tmp(3) tmp(3) tmp(4) tmp(4) tmp(3)];
      line(x,y,'Color','r')
      if sub_rois_del(i)
        xd=[tmp(1) tmp(2) tmp(2) tmp(1)];
        yd=[tmp(3) tmp(4) tmp(3) tmp(4)];
        line(xd,yd,'Color','r')
      end
    end
    drawnow
    title('Sub-ROI','Interpreter','none')
    zoom on
```

```
case ('2D NPS')
  if ~isfield(handles,'sub_rois_fft2d')
    h=msgbox({'2D NPS of Sub-ROIs not computed yet!'},...
      'Instruction','help','modal');
    waitfor(h)
    return
  end
  plot_bar(handles,handles.sub_rois_fft2d);
  axis off;
  title('2D NPS');

case ('NPS HZ+VT+45')
     x_size = handles.ROISize;
     x_axis_start = (x_size/2);

     lp_hz = 1/(handles.strXdel*2);
     lp_hz_axis = lp_hz*(2:x_axis_start)/x_axis_start;

     lp_vt = 1/(handles.strYdel*2);
     lp_vt_axis = lp_vt*(2:x_axis_start)/x_axis_start;

     lp_45_axis=sqrt(power(lp_hz,2)+power(lp_vt,2))*(3:x_axis_start)/x_axis_start;


  if isfield(handles,'nps_hz')
    hold on
    if (get(handles.logplot,'Value')) %handles.logplot
      warning off;
      plot(lp_hz_axis,log10(handles.nps_hz(x_axis_start+2:x_size)),'r.-')
      plot(lp_vt_axis,log10(handles.nps_vt(x_axis_start+2:x_size)),'b.-')
      plot(lp_45_axis,log10(handles.nps_45),'g.-')
      warning on;
      xlabel('Spatial Frequency (lp/mm)','FontSize',14)
      ylabel('NPS (mm^2)', 'FontSize',14)
    else
      plot(lp_hz_axis,handles.nps_hz(x_axis_start+2:x_size),'r.-')
      plot(lp_vt_axis,handles.nps_vt(x_axis_start+2:x_size),'b.-')
      plot(lp_45_axis,handles.nps_45,'g.-')
     xlabel('Spatial Frequency (lp/mm)', 'FontSize',10)
      ylabel('NPS (mm^2)', 'FontSize',10)
    end
    legend({'Horizontal', 'Vertical', '45 Degree'})
    zoom on
  end
case ('NPS Horizontal')
     x_size = handles.ROISize;
     x_axis_start = x_size/2;

     lp_hz = 1/(handles.strXdel*2);
     lp_hz_axis = lp_hz*(2:x_axis_start)/x_axis_start;

     lp_vt = 1/(handles.strYdel*2);
     lp_vt_axis = lp_vt*(2:x_axis_start)/x_axis_start;
```

```
        lp_45_axis=sqrt(power(lp_hz,2)+power(lp_vt,2))*(3:x_axis_start)/x_axis_start;

      if isfield(handles,'nps_hz')
         hold on
        if (get(handles.logplot,'Value')) %handles.logplot
          warning off;
          plot(lp_hz_axis,log10(handles.nps_hz(x_axis_start+2:x_size)),'r.-')
          xlabel('Spatial Frequency (lp/mm)')
          ylabel('NPS (mm^2)')
          warning on;
        else
          plot(lp_hz_axis,handles.nps_hz(x_axis_start+2:x_size),'r.-')
          xlabel('Spatial Frequency (lp/mm)')
          ylabel('NPS (mm^2)')
        end
        zoom on
      end
    case ('NPS Vertical')
        x_size = handles.ROISize;
        x_axis_start = x_size/2;

        lp_hz = 1/(handles.strXdel*2);
        lp_hz_axis = lp_hz*(2:x_axis_start)/x_axis_start;

        lp_vt = 1/(handles.strYdel*2);
        lp_vt_axis = lp_vt*(2:x_axis_start)/x_axis_start;

        lp_45_axis=sqrt(power(lp_hz,2)+power(lp_vt,2))*(3:x_axis_start)/x_axis_start;

      if isfield(handles,'nps_hz')
        hold on
        if (get(handles.logplot,'Value')) %handles.logplot
          warning off;
          plot(lp_vt_axis,log10(handles.nps_vt(x_axis_start+2:x_size)),'b.-')
          xlabel('Spatial Frequency (lp/mm)')
          ylabel('NPS (mm^2)')
          warning on;
        else
          plot(lp_vt_axis,handles.nps_vt(x_axis_start+2:x_size),'b.-')
          xlabel('Spatial Frequency (lp/mm)')
          ylabel('NPS (mm^2)')
        end
        zoom on
      end
    case ('NPS 45')
        x_size = handles.ROISize;
        x_axis_start = x_size/2;

        lp_hz = 1/(handles.strXdel*2);
        lp_hz_axis = lp_hz*(3:x_axis_start)/x_axis_start;

        lp_vt = 1/(handles.strYdel*2);
        lp_vt_axis = lp_vt*(3:x_axis_start)/x_axis_start;
```

```
        lp_45_axis=sqrt(power(lp_hz,2)+power(lp_vt,2))*(3:x_axis_start)/x_axis_start;


       if isfield(handles,'nps_hz')
         hold on
         if (get(handles.logplot,'Value')) %handles.logplot
           warning off;
           plot(lp_45_axis,log10(handles.nps_45),'g.-')
           xlabel('Spatial Frequency (lp/mm)')
           ylabel('NPS (mm^2)')
           warning on;
           handles.shown_img = plot(lp_45_axis,log10(handles.nps_45),'g.-');
           guidata(handles.nps_tool,handles)
         else
           plot(lp_45_axis,handles.nps_45,'g.-')
           xlabel('Spatial Frequency (lp/mm)')
           ylabel('NPS (mm^2)')
         end
         zoom on
       end

    otherwise
      return
  end

  zoom reset

%%--------------------------------------------------------------------
%% plot imgage, reset colorbar and figure size. matlab default was too big
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plot_bar(handles,img)
  % plot_bar returns h_img, a imagesc graphic object
  %               h_plot, a plot graphic object

  %handles.shown_img = img;
  guidata(handles.nps_tool,handles)

  h_img=-1;
  h_plot=-1;
%   if ~(handles.roi_state)
%       return
%   end
   %desinate ploting 'window'/axes
   axes(handles.axes1);
   cla; %clear the axes for plotting


  figpos=handles.axes1_pos;
  %position = [left bottom width height];
   set(handles.axes1, ...
     'Position',[figpos(1) figpos(2) ...
       figpos(3)*.96 figpos(4)],...
      'FontSize',9);
```

```
%          figpos(3)*0.96 figpos(4)],...
%         'FontSize',9);


   if (get(handles.logplot,'Value')) %handles.logplot
       warning off;
       h_img=imagesc(log10(double(abs(img(:,:,1)))));
       warning on;
   else
       h_img=imagesc(img(:,:,1));
   end

    %use img2 for plot command only
    if exist('img2','var')
      hold on
      h_plot=plot(img2,'rx');
      hold off
   end



   h_bar=colorbar;%('v6'); %('East');

   %axes(h_bar);
   %cla;
    %h_bar = findobj(gcf,'Tag','Colorbar')
%   barpos = get(h_bar,'Position');
    set(h_bar, ...
      'Position',[figpos(3)+.025 figpos(2) ...
         .01 figpos(4)], ...
      'FontSize',7);%,'tickdir','out');

   zoom on


% --- Executes during object creation, after setting all properties.
function plot_CreateFcn(hObject, eventdata, handles)
% hObject    handle to plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in logplot.
function logplot_Callback(hObject, eventdata, handles)
% Hint: get(hObject,'Value') returns toggle state of logplot

%   if ~(handles.roi_state) || ~isfield(handles,'img')
%       return
```

```
%    end
    plot_obj=findall(gcf,'Tag','plot');
    plot_Callback(plot_obj,eventdata, handles)



% --- Executes on button press in Choose_roi.
function Choose_roi_Callback(hObject, eventdata, handles, coord)

    if ~isfield(handles,'img') %|| ~(handles.roi_state) %|| ~isfield(handles,'img')
                h=msgbox({'Image has not been loaded!'},...
              'Instruction','help','modal');
            waitfor(h)


       return
    end


   plot_obj=findall(gcf,'Tag','plot');
   contents = get(plot_obj,'String');

   img_roi=(handles.img(:,:,1));
   coord_orig=handles.img_orig_coord;

   switch contents{get(plot_obj,'Value')}
     case ('ROI image')

        img_roi=(handles.img_roi);
        coord_orig=handles.img_roi_axis;

      otherwise
        handles.roi_state=1;
        guidata(hObject,handles)
        set(plot_obj,'Value',...
          strmatch({'original image'},get(plot_obj,'String'),'exact'));
        plot_Callback(plot_obj,eventdata, handles)
        handles.roi_state=0;

        handles.track.rot90=0;
        handles.track.flipud=0;
        handles.track.fliplr=0;

        guidata(hObject,handles)
   end

    %check if called by batch
    if ~exist('coord','var')
       %H = NPS_ROI_OPTIONS({[x1 x2 y1 y2]},{[sub_roi_size]}) takes the coordinates of the original
size of image,
       %    returns user's choice of ROI tool. When user choses to specify ROI,
       %    coordinates will be returned. Second element in the input cell
```

```
      %    array is the default sub-roi size
      %H = 1 by 3 cell array. ({'string', [x1 x2 y1 y2],[sub_roi_size]})
      %    first element = choice of ROI tool ('rect marquee' or 'coordinates').
      %    second element = coordiantes for specified ROI [x1 x2 y1 y2]. [] is returned when
      %      rectangular marguee is the choice.
      %    thrid element = size of sub-roi, recommended size = 128. (128x128)
      %      must a power of 2 value.
     option=nps_ROI_options({coord_orig},{[100]});

     if isempty(option)
       if isfield(handles,'track') && isfield(handles.track,'roi_coord')
         handles.track=rmfield(handles.track,'roi_coord');
       end
       handles.roi_state=1;
       guidata(hObject,handles)
       return
     end

     if strmatch(option{1}, 'rect marquee')
       zoom on

       h=msgbox({'Drag left mouse button to select ROI',...
            'Press "Set Roi" button above the image when done'}, 'Instruction','help','modal');

       set(handles.set_roi,'visible','on')

       waitfor(handles.set_roi,'visible','off')
       zoom off

       % Extract the full region selected to be used ;
       axis_img=round(axis);
       axis_img(find(axis_img<1))=1;
       axis_img(find(axis_img(1:2)>size(img_roi,2)))=size(img_roi,2);
       axis_img(find(axis_img(3:4)>size(img_roi,1))+2)=size(img_roi,1);
       axis_img;

%
%
     elseif strmatch(option{1},'coordinates')
       axis_img=option{2};
     end
   else %if called by batch
     axis_img=coord;
   end


   %extract ROI now
%    img_roi=img_roi(axis_img(3):axis_img(4),...
%      axis_img(1):axis_img(2));

   % keep track
   handles.track.roi_coord=axis_img;
   handles.img_roi_axis=axis_img;
   img_roi=img_roi/max(max(img_roi));
```

```
    handles.img_roi=img_roi;
    handles.roi_state=1;
    guidata(hObject,handles)

%     %compute subroi coordinates
%     sub_roi_size=option{3};
%     subroi_coord(sub_roi_size,coord_orig,handles)
%
%     handles=guidata(handles.nps_tool);
%
    set(plot_obj,'Value',...
      strmatch({'ROI image'},get(plot_obj,'String'),'exact'))
%       strmatch({'Sub-ROI on ROI img'},get(plot_obj,'String'),'exact'))
    plot_Callback(plot_obj,eventdata, handles)


% --- Executes on button press in SUB_ROI.
function SUB_ROI_Callback(hObject, eventdata, handles)
% hObject    handle to SUB_ROI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
  if ~isfield(handles,'img_roi_axis') %|| ~(handles.roi_state) %|| ~isfield(handles,'img')
     return
  end

    %compute subroi coordinates
    sub_roi_size=Sub_ROI_size({1});
%    sub_roi_size=Sub_ROI_size({128}); Original
    coord_orig=handles.img_orig_coord;


    subroi_coord(sub_roi_size{1},coord_orig,handles)

    handles=guidata(handles.nps_tool);

    set(handles.plot,'Value',...
       strmatch({'Sub-ROI on ROI img'},get(handles.plot,'String'),'exact'))
%       strmatch({'ROI image'},get(plot_obj,'String'),'exact'))
    plot_Callback(handles.plot,eventdata, handles)


% --- compute subroi coordinates
function subroi_coord(sub_roi_size,coord_orig,handles)

  roi_coords=handles.img_roi_axis;%handles.track.roi_coord;
  img_roi=handles.img_roi;

  handles.ROISize = sub_roi_size;



  if find(size(img_roi)<(sub_roi_size)./2)
```

```
    h=msgbox({'ROI must be larger than sub-roi size for nps calculation!'},...
      'Instruction','help','modal');
    waitfor(h)
    handles=rmfield(handles,'sub_rois');
    handles=rmfield(handles,'sub_rois_del');
    handles=rmfield(handles,'sub_rois_axis');
    guidata(handles.nps_tool,handles)
    return
  end

  %row
  row_size=round((roi_coords(4)-roi_coords(3)+1)/sub_roi_size);
  if ((row_size*sub_roi_size)+roi_coords(3)-1) > size(handles.img,1)
    row_size=row_size-1;
  end
  %column
  col_size=round((roi_coords(2)-roi_coords(1)+1)/sub_roi_size);
  if ((col_size*sub_roi_size)+roi_coords(1)-1) > size(handles.img,2)
    col_size=col_size-1;
  end



  %sub_roi
  count=1;
  for i = 1:row_size
    for j=1:col_size
      %x1 x2 y1 y2
      sub_rois{count}=[ ((j-1)*sub_roi_size)+roi_coords(1) (j*sub_roi_size)+roi_coords(1)-1 ...
        ((i-1)*sub_roi_size)+roi_coords(3) (i*sub_roi_size)+roi_coords(3)-1];% +...
        %[coord_orig(1) coord_orig(1) coord_orig(3) coord_orig(3)];
      count=count+1;
    end
  end

  %coordinates of the boundary of all sub-ROIs
  sub_rois_axis=[roi_coords(1) (sub_roi_size*col_size)+roi_coords(1)-1 ...
    roi_coords(3) (sub_roi_size*row_size)+roi_coords(3)-1];

  handles.sub_rois_axis=sub_rois_axis;
  %img_roi_axis = sub_roi_axis
  handles.img_roi_axis=sub_rois_axis;
  handles.sub_rois=sub_rois;
  handles.sub_rois_del=zeros(size(sub_rois));
  guidata(handles.nps_tool,handles)


% --- Executes on selection change in color_map.
function color_map_Callback(hObject, eventdata, handles)
  %if ~(handles.roi_state)
  %    return
  %end

  contents = get(hObject,'String');
  axes(handles.axes1);
```

```
 cm=contents{get(hObject,'Value')};
 colormap(char(cm))

% --- Executes during object creation, after setting all properties.
function color_map_CreateFcn(hObject, eventdata, handles)
% hObject    handle to color_map (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------
function Batch_Callback(hObject, eventdata, handles)
% hObject    handle to Batch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 if ~exist('file_tr','var') %filename') && ~exist('pathname')

    %gui for erasing old data
    if isfield(handles,'img')
       button=questdlg({'All data will be lost', 'Proceed?'},...
          'Open new image','Yes','No','Yes');
       if strcmp(button,'No')
          return
       end
    end


    %reset handles (erase old data)
    handles=handles.start_condition;
    handles.start_condition=handles;
    guidata(hObject,handles);
    %size(fieldnames(handles.start_condition))


%    [filename, pathname] = uigetfiles( ...
%     ['*.bmp;*.gif;*.jpg;*.pgm;*.ras;*.tif;*.dcm;*.raw;*.mat'], ...
%     'Pick a file')

  [filename, pathname, filterindex] = uigetfile( ...
       {'*.bmp;*.gif;*.jpg;*.pgm;*.ras;*.tif;*.dcm;*.raw', ...
       'All Image Formats (*.bmp;*.gif;*.jpg;*.pbm;*.pgm;*.png;*.ras;*.tif;*.dcm;*.raw)'; ...
       '*.bmp', 'Windows Bitmap (*.bmp)'; ...
       '*.gif', 'Graphics Interchange Format (*.gif)'; ...
       '*.jpg', 'Joint Photographic Experts Group (*.jpg)'; ...
       '*.pgm', 'Portable graymap (*.pgm)'; ...
       '*.ras', 'Sun Raster File (*.ras)'; ...
       '*.tif', 'Tagged Image File Format (*.tif)'; ...
       '*.dcm', 'Dicom Image File Format (*.dcm)'; ...
```

```
        '*.raw', 'RAW Image File Format (*.raw)'; ...
        '*.*',  'All Files (*.*)'}, ...
        'Pick a file','MultiSelect', 'on');

   else
     %filename returned from fileparts does not have extension!!
     [pathname,filename,ext] = fileparts(file_tr);
     filename=[filename ext];
     filterindex=1;
  end

% Loads the image files into a 3d matrix img
i = 1;

while i < length(filename) +1
files = char(filename(i))


   handles.filename=filename;
   handles.pathname=pathname;
   handles.number_images = length(filename);
   guidata(hObject, handles);

   if filterindex~=0
     [pathstr,name,ext] = fileparts(files);
     switch ext
        case '.dcm'
           img(:,:,i)=dicomread(fullfile(pathname,files));%[pathname filename]);
        case '.raw'
           if ~exist('file_tr','var') || ~isfield(handles,'track')
              [img,raw_opt]=raw_hk({[fullfile(pathname,filename)]});
           elseif isfield(handles.track,'raw_opt')
              [img(:,:,i),raw_opt]=raw_hk({[fullfile(pathname,files)]},...
                 {handles.track.raw_opt});
           else
              h=msgbox({'Raw image file not opened!!'},...
                 'Message','help','modal');
              waitfor(h)
              return
           end
           handles.track.raw_opt=raw_opt;
        otherwise
           [img(:,:,i),map] = imread(fullfile(pathname,files));%[pathname filename]);
     end
  else
     axes(handles.axes1);
     cla; % reset; %clear the axes for plotting
     return;
   end

 % Save the image information into a 3d matrix handle.img
  handles.img(:,:,i)=img(:,:,i);
  i =i+1;
end
```

```
% OPENS ONE OF THE IMAGES THAT WAS SELECTED
handles.img_roi=double(img(:,:,1))/max(max(double(img(:,:,1))));
%%matlab size command returns -rows*columns
%%axis return [xmin xmax ymin ymax]
img_roi_axis=[1 size(img(:,:,1),2) 1 size(img(:,:,1),1)];
handles.img_orig_coord=img_roi_axis;%boundary of orig image, bascially the size
handles.img_roi_axis=img_roi_axis;%axis coord of roi on ROI image
handles.img_roi_axis_orig=img_roi_axis; %axis coord of roi on original image
handles.filename=files;
handles.track=[]; %tracking structure
handles.batch_status = 1; %needed to determining the NPS for the batched images

guidata(hObject,handles);

plot_obj=handles.plot;%findall(gcf,'Tag','plot');
set(plot_obj,'Value',...
   strmatch({'original image'},get(plot_obj,'String'),'exact'))
plot_Callback(plot_obj,eventdata, handles)

%colormap gray
color_obj=handles.color_map;%findall(gcf,'Tag','color_map');
color_map_Callback(color_obj, eventdata, handles);

% --------------------------------------------------------------------
function Open_Callback(hObject, eventdata, handles, file_tr)
handles.roi_state=0;
guidata(hObject,handles);
%  if ~(handles.roi_state)
%     return
%  end

% file_tr containing tracking information
if ~exist('file_tr','var') %filename') && ~exist('pathname')

   %gui for erasing old data
   if isfield(handles,'img')
      button=questdlg({'All data will be lost', 'Proceed?'},...
         'Open new image','Yes','No','Yes');
      if strcmp(button,'No')
         return
      end
   end

   %reset handles (erase old data)
   handles=handles.start_condition;
   handles.start_condition=handles;
   guidata(hObject,handles);
   %size(fieldnames(handles.start_condition))

   [filename, pathname, filterindex] = uigetfile( ...
      {'*.bmp;*.gif;*.jpg;*.pgm;*.ras;*.tif;*.dcm;*.raw', ...
      'All Image Formats (*.bmp;*.gif;*.jpg;*.pbm;*.pgm;*.png;*.ras;*.tif;*.dcm;*.raw)'; ...
      '*.bmp', 'Windows Bitmap (*.bmp)'; ...
```

```
        '*.gif', 'Graphics Interchange Format (*.gif)'; ...
        '*.jpg', 'Joint Photographic Experts Group (*.jpg)'; ...
        '*.pgm', 'Portable graymap (*.pgm)'; ...
        '*.ras', 'Sun Raster File (*.ras)'; ...
        '*.tif', 'Tagged Image File Format (*.tif)'; ...
        '*.dcm', 'Dicom Image File Format (*.dcm)'; ...
        '*.raw', 'RAW Image File Format (*.raw)'; ...
        '*.*',  'All Files (*.*)'}, ...
        'Pick a file');
else
    %filename returned from fileparts does not have extension!!
    [pathname,filename,ext] = fileparts(file_tr);
    filename=[filename ext];
    filterindex=1;
end

if filterindex~=0
    [pathstr,name,ext] = fileparts(filename);
    switch ext
        case '.dcm'
            img=dicomread(fullfile(pathname,filename));%[pathname filename]);
        case '.raw'
            if ~exist('file_tr','var') || ~isfield(handles,'track')
                [img,raw_opt]=raw_hk({[fullfile(pathname,filename)]});
            elseif isfield(handles.track,'raw_opt')
                [img,raw_opt]=raw_hk({[fullfile(pathname,filename)]},...
                    {handles.track.raw_opt});
            else
                h=msgbox({'Raw image file not opened!!'},...
                    'Message','help','modal');
                waitfor(h)
                return
            end
            handles.track.raw_opt=raw_opt;
        otherwise
            [img,map] = imread(fullfile(pathname,filename));%[pathname filename]);
    end
else
    axes(handles.axes1);
    cla; % reset; %clear the axes for plotting
    return;
end

warning off
if ndims(img)>2 %~isgray(img)
    h=msgbox({'The image is not grayscale '},...
        'Message','help','modal');
    waitfor(h)
    return
end

% erase old data if failed to open image
if isempty(img) || ~any(size(img))
    h=msgbox({'Image file not opened!!'},...
        'Message','help','modal');
```

```
    waitfor(h)
    if isfield(handles,'img')
        handles=rmfield(handles,'img');
    end
    handles=handles.start_condition;
    handles.start_condition=handles;
    guidata(hObject,handles);
    return
end

set(handles.nps_tool,'Name',filename)

warning on
%img=double(img);
handles.img=img;

handles.img_roi=double(img)/max(max(double(img)));
%%matlab size command returns -rows*columns
%%axis return [xmin xmax ymin ymax]
img_roi_axis=[1 size(img,2) 1 size(img,1)];
handles.img_orig_coord=img_roi_axis;%boundary of orig image, bascially the size
handles.img_roi_axis=img_roi_axis;%axis coord of roi on ROI image
handles.img_roi_axis_orig=img_roi_axis; %axis coord of roi on original image
handles.filename=filename;
handles.track=[]; %tracking structure
handles.batch_status = 0; %needed to determining the NPS for the non-batched image

guidata(hObject,handles);

plot_obj=handles.plot;%findall(gcf,'Tag','plot');
set(plot_obj,'Value',...
    strmatch({'original image'},get(plot_obj,'String'),'exact'))
plot_Callback(plot_obj,eventdata, handles)

%colormap gray
color_obj=handles.color_map;%findall(gcf,'Tag','color_map');
color_map_Callback(color_obj, eventdata, handles);




% Opens MATLAB .mat files
% --------------------------------------------------------------------
function matlab_bin_Callback(hObject, eventdata, handles, file_tr)



 %check if it's batch call
 if ~exist('file_tr','var') %filename') && ~exist('pathname')
    h=msgbox({'The image data should be assigned to ','the ONLY variable in the .mat file.'},...
      'Message','help','modal');
    waitfor(h)
```

```
  %gui for data erase
  if isfield(handles,'img')
    button=questdlg({'All data will be lost', 'Proceed?'},...
      'Open new image','Yes','No','Yes');
    if strcmp(button,'No')
      return
    end
  end
  %reset handles
  handles=handles.start_condition;
  handles.start_condition=handles;

  [filename, pathname, filterindex] = uigetfile( {'*.mat'}, 'Pick a file');
else
  %filename returned from fileparts does not have extension!!
  [pathname,filename,ext] = fileparts(file_tr);
  filename=[filename ext];
  filterindex=1;
end

if (filterindex~=0)
  %% using load in this envirn will return a struct
  a=load(fullfile(pathname,filename));
  set(handles.nps_tool,'Name',filename)
else
  axes(handles.axes1);
  cla;% reset; %clear the axes for plotting
  return;
end

 tmp=fieldnames(a);
 eval(['img=a.' char(tmp(1)) ';'])

% erase old data if failed to open image
if isempty(img) || ~any(size(img))
  h=msgbox({'Image file not opened!!'},...
    'Message','help','modal');
  waitfor(h)
  if isfield(handles,'img')
    handles=rmfield(handles,'img');
  end
  handles=handles.start_condition;
  handles.start_condition=handles;
  guidata(hObject,handles);
  return
end

 handles.img=img;
 handles.img_roi=double(img)/max(max(double(img)));

 img_roi_axis=[1 size(img,2) 1 size(img,1)];
handles.img_orig_coord=img_roi_axis;%boundary of orig image, bascially the size
handles.img_roi_axis=img_roi_axis;%axis coord of roi on ROI image
handles.img_roi_axis_orig=img_roi_axis; %axis coord of roi on original image
```

```
handles.filename=filename;
handles.track=[]; %tracking structure
handles.batch_status = 0; %needed to determining the NPS for the non-batched image
guidata(hObject,handles);

plot_obj=findall(gcf,'Tag','plot');
set(plot_obj,'Value',...
   strmatch({'original image'},get(plot_obj,'String')))
plot_Callback(plot_obj,eventdata, handles)

%colormap gray
color_obj=handles.color_map;%findall(gcf,'Tag','color_map');
color_map_Callback(color_obj, eventdata, handles);


% --------------------------------------------------------------------
function save_img_bin_Callback(hObject, eventdata, handles)
if ~isfield(handles,'img_roi') %|| ~(handles.roi_state) %|| ~isfield(handles,'img_roi')
   return
end

[FileName,PathName,FilterIndex] = uiputfile('*.mat','Enter filename');

if FilterIndex~=0
   img=handles.img_roi;
   save(fullfile(PathName,FileName),'img');
else
   return;
end



% --------------------------------------------------------------------
function roi_img_tiff_Callback(hObject, eventdata, handles)
if ~isfield(handles,'img_roi') %|| ~(handles.roi_state) %|| ~isfield(handles,'img_roi')
   return
end

[FileName,PathName,FilterIndex] = uiputfile('*.tif','Enter filename');

if FilterIndex~=0
   img=uint16(round(handles.img_roi.*(2^16-1)));
   imwrite(img,fullfile(PathName,FileName),'tif');
else
   return;
end



% --------------------------------------------------------------------
function FFT2D_save_Callback(hObject, eventdata, handles)
%   if ~(handles.roi_state)
```

```
%     return
%   end
  if ~isfield(handles,'sub_rois_fft2d')
    h=msgbox({'2D NPS of Sub-ROIs not computed yet.'},...
      'Instruction','help','modal');
    waitfor(h)
    return
  end
  [FileName,PathName,FilterIndex] = uiputfile('*.mat','Enter filename');

  if FilterIndex~=0
    img=handles.sub_rois_fft2d;
    save(fullfile(PathName,FileName),'img');
  else
    return;
  end

  % --------------------------------------------------------------------
function Save_nps_Callback(hObject, eventdata, handles)
%   if ~(handles.roi_state)
%     return
%   end
  if ~isfield(handles,'nps_hz')
    h=msgbox({'NPS horizontal and vertical not computed yet!'},...
      'Instruction','help','modal');
    waitfor(h)
    return
  end
  [FileName,PathName,FilterIndex] = uiputfile('*.mat','Enter filename');

  if FilterIndex~=0
    nps_hz=handles.nps_hz;
    nps_vt=handles.nps_vt;
    nps_45=handles.nps_45;
    nps_num_lines=handles.nps_num_lines;
    save(fullfile(PathName,FileName),'nps_hz','nps_vt','nps_45','nps_num_lines');
  else
    return;
  end



% --------------------------------------------------------------------
function print_current_figure_Callback(hObject, eventdata, handles)
% hObject    handle to print_current_figure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(gcf,'PaperPositionMode','auto')
printpreview()


% --------------------------------------------------------------------
function fig2file_Callback(hObject, eventdata, handles)
% hObject    handle to fig2file (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

print -djpeg -r300 Current_Figure.jpg

% --------------------------------------------------------------------
function File_Callback(hObject, eventdata, handles)
% hObject    handle to File (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function save_menu_Callback(hObject, eventdata, handles)
% hObject    handle to save_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function print_menu_Callback(hObject, eventdata, handles)
% hObject    handle to print_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in set_roi.
function set_roi_Callback(hObject, eventdata, handles)
   set(handles.set_roi,'visible','off')

% ------------------------------------------------------------
% --- Executes on button press in Linearization.
function linearization_Callback(hObject, eventdata, handles)
   if ~isfield(handles,'img_roi') %|| ~(handles.roi_state) %|| ~isfield(handles,'img_roi')
     return
   end
   h=msgbox({'Linearization must be performed before choosing ROI'},...
     'Instruction','help','modal');
   waitfor(h)
   %call gui
   v=linearization_gui;
   %use original image
   img=double(handles.img);

   switch v
     case 'image = image .^2'
       %handles.img_roi=img.^2;
       handles.img=img.^2;
       guidata(hObject,handles);
     case 'Fuji log compression'
       prompt = {'Latitude:'...
         'Sensitivity'};
       dlg_title = 'Fuji Log Compression';
       num_lines=[1,40];
```

```
        def = {'2','96'};
        options.Resize='on';
        options.WindowStyle='modal';
        options.Interpreter='tex';
        answer  = inputdlg(prompt,dlg_title,num_lines,def,options);
        if isempty(answer)
          return
        end
        lat=str2num(answer{1});
        sens=str2num(answer{2});
        sk=4-log10(sens/4);
        handles.img=10.^((img-511+1023/lat*sk)*lat/1023);
        guidata(hObject,handles);
      otherwise
        h=msgbox({'Linearization not performed'}, 'Instruction','help','modal');
        waitfor(h)
        return
    end

    plot_obj=findall(gcf,'Tag','plot');
    set(plot_obj,'Value',...
      strmatch({'ROI image'},get(plot_obj,'String'),'exact'))
    plot_Callback(plot_obj,eventdata, handles)




% --- Executes on button press in sub_roi_done.
function sub_roi_done_Callback(hObject, eventdata, handles)
% hObject    handle to sub_roi_done (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    set(handles.sub_roi_done,'visible','off')
    handles.sub_roi_rm=0;
    guidata(handles.delete_sub_roi,handles);



% --- Executes on button press in delete_sub_roi.
function delete_sub_roi_Callback(hObject, eventdata, handles)
% hObject    handle to delete_sub_roi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    if ~isfield(handles,'img')  || ~isfield(handles,'sub_rois')
      return
    end

    set(handles.sub_roi_done,'visible','on')
    handles.sub_roi_rm=1;
    guidata(handles.delete_sub_roi,handles);

    set(handles.plot,'Value',...
       strmatch({'Sub-ROI on ROI img'},get(handles.plot,'String'),'exact'))
    plot_Callback(handles.plot,eventdata, handles)
```

```
h=msgbox({'Click on the sub-roi to be deleted,',...
  'Click again to remove deletion.'...
  'DO NOT drag the mouse while clicking',...
  'Press "Done" button above the image when finished'}, 'Instruction','help','modal');
waitfor(h)

%set pointer style to arrow
zoom off
set(handles.nps_tool,'Pointer','arrow')
x=[];y=[];

img=handles.img;
img_x=size(img,1); %row
img_y=size(img,2); %column

sub_rois_del=handles.sub_rois_del;
sub_rois=handles.sub_rois;

% 'CurrentPoint' returns [x y] of the last mouse button down position
set(handles.nps_tool,'units','pixels')
pt_ex = get(handles.axes1, 'CurrentPoint');

while handles.sub_roi_rm

  pt_now = get(handles.axes1, 'CurrentPoint');
  pt_x=pt_now(1,1);pt_y=pt_now(1,2);

  if ~isequal(pt_ex,pt_now) && all([pt_x pt_y] > 0) ...
      && all([pt_x pt_y] < [img_x img_y])

    % check clicked position
    for i =1 : size(sub_rois,2)
      if pt_x >= sub_rois{i}(1) && pt_x <= sub_rois{i}(2)
        if pt_y >= sub_rois{i}(3) && pt_y <= sub_rois{i}(4)
          sub_rois_del(i)=abs(sub_rois_del(i)-1);

          handles.sub_rois_del=sub_rois_del;
          plot_Callback(handles.plot,eventdata, handles)
          zoom off
          set(handles.nps_tool,'Pointer','arrow')

        end
      end
    end % for i ...

  end %~isequal ...

  %get updated handles check for 'done' button click
  handles=guidata(handles.delete_sub_roi);
  pause(.05)
  pt_ex=pt_now;

end %while handles ...
```

```
    handles.sub_rois_del=sub_rois_del;
    guidata(hObject,handles);
    zoom on


%--- Updates the x smpling pitch
function varargout = strXdel_Callback(hObject, eventdata, handles, varargin)

NewVal = str2double(get(hObject,'String'));
handles.strXdel = NewVal;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function strXdel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to strXdel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



%--- Updates the y smpling pitch

function strYdel_Callback(hObject, eventdata, handles)

NewVal = str2double(get(hObject,'String'));
handles.strYdel = NewVal;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function strYdel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to strYdel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in FFT2d.
```

```
function FFT2d_Callback(hObject, eventdata, handles)

   if ~isfield(handles,'img')
    h=msgbox({'An image has not been loaded yet'},...
      'Instruction','help','modal');
    waitfor(h)
     return
   end

   if ~isfield(handles,'mean')  %check to see if mean type has been selected
     handles.mean = 0;
     guidata(hObject,handles);
  end



x_del = handles.strXdel;
y_del = handles.strYdel;
mean_status =  handles.mean;

%   Calculates NPS if no ROI and sub-sampling areas have been selected
if handles.roi_state==0 && ~isfield(handles,'sub_rois')
          h=msgbox({'NO ROI selected'},...
       'Instruction','help','modal');
    waitfor(h)

img=handles.img;
Size = size(img,1).*size(img,2);

      if handles.mean == 2  % Calculates "No mean method used"

       fft2d=(abs(power(fftshift(fft2(double(img),size(img,1),size(img,2))),2)))*((x_del*y_del)/Size);

       handles.sub_rois_fft2d=fft2d;%sub_rois_fft2d;
       guidata(hObject,handles);

       set(handles.plot,'Value',...
       strmatch({'2D NPS'},get(handles.plot,'String'),'exact'))
       plot_Callback(handles.plot,eventdata, handles)
       end

        if handles.mean == 1  % Calculates "Local Mean"
                h=msgbox({'Not able to calculate because no sub-ROI region was selected'},...
       'Instruction','help','modal');
      waitfor(h)
      return
        end

        if handles.mean == 0  % Calculates "Grand Mean"
     roi_mean=mean(mean(img));
      img=img-roi_mean;
      fft2d=(abs(power(fftshift(fft2(double(img))),2)))*((x_del*y_del)/Size);
```

```
        handles.sub_rois_fft2d=fft2d;%sub_rois_fft2d;
        guidata(hObject,handles);

        set(handles.plot,'Value',...
        strmatch({'2D NPS'},get(handles.plot,'String'),'exact'))
        plot_Callback(handles.plot,eventdata, handles)
         end
        return
end

%   Calculates NPS if ROI is slected and No sub-sampling areas have been selected
if handles.roi_state==1  && ~isfield(handles,'sub_rois')
            h=msgbox({'ROI slected, but No sub-ROI Sampling area selected'},...
         'Instruction','help','modal');
       waitfor(h)
        img=handles.img;

        c=handles.img_roi_axis;
        roi_mean=mean(mean(img(c(3):c(4),c(1):c(2))));
        img(c(3):c(4),c(1):c(2))=img(c(3):c(4),c(1):c(2))-roi_mean;

        Size = size(img(c(3):c(4),c(1):c(2)),1).*size(img(c(3):c(4),c(1):c(2)),2);
        fft2d=(abs(power(fftshift(fft2(double(img(c(3):c(4),c(1):c(2))))),2)))*((x_del*y_del)/Size);

        handles.sub_rois_fft2d=fft2d;%sub_rois_fft2d;
        guidata(hObject,handles);

        set(handles.plot,'Value',...
        strmatch({'2D NPS'},get(handles.plot,'String'),'exact'))
        plot_Callback(handles.plot,eventdata, handles)

  return
 end

if handles.batch_status == 0  % Checks if it is a single image

       Size = handles.ROISize.*handles.ROISize;
       sub_rois_del=handles.sub_rois_del;
       sub_rois=handles.sub_rois;
       img=handles.img;


   if ~isequal(size(sub_rois_del),size(sub_rois))
     h=msgbox({'Sub-ROI slection error in function subroi_avg_Callback'},...
       'Instruction','help','modal');
     waitfor(h)
   else

    if handles.mean == 2  % Calculates "No mean method used"

      c=handles.sub_rois_axis;
```

```
 c=sub_rois{1};
 fft2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1);

 count=0;

 for i=1:size(sub_rois,2)
   if ~sub_rois_del(i)
     c=sub_rois{i};
     fft2d=fft2d+(abs(power(fftshift(fft2(double(img(c(3):c(4),c(1):c(2))))),2)))*((x_del*y_del)/Size);
     count=count+1;

   end
 end
 fft2d=fft2d./count;
end

 if handles.mean == 0  % Calculates "grand mean"

 c=handles.sub_rois_axis;
 roi_mean=mean(mean(img(c(3):c(4),c(1):c(2))));
 img(c(3):c(4),c(1):c(2))=img(c(3):c(4),c(1):c(2))-roi_mean;


 c=sub_rois{1};
 fft2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1);

 count=0;

 for i=1:size(sub_rois,2)
   if ~sub_rois_del(i)
     c=sub_rois{i};
     fft2d=fft2d+(abs(power(fftshift(fft2(double(img(c(3):c(4),c(1):c(2))))),2)))*((x_del*y_del)/Size);
     count=count+1;

   end
 end
 fft2d=fft2d./count;
end

if handles.mean == 1  % Calculates "local mean"

 c=handles.sub_rois_axis;
 c=sub_rois{1};
 fft2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1);

 count=0;

 for i=1:size(sub_rois,2)
   if ~sub_rois_del(i)
     c=sub_rois{i};
     roi_mean=mean(mean(img(c(3):c(4),c(1):c(2))));
     img(c(3):c(4),c(1):c(2))=img(c(3):c(4),c(1):c(2))-roi_mean;
     fft2d=fft2d+(abs(power(fftshift(fft2(double(img(c(3):c(4),c(1):c(2))))),2)))*((x_del*y_del)/Size);
```

```
            count=count+1;

          end
        end
        fft2d=fft2d./count;
      end



    end

    handles.sub_rois_fft2d=fft2d;%sub_rois_fft2d;
    guidata(hObject,handles);

end

if handles.batch_status == 1  % Checks if it is batch images
        Size = handles.ROISize.*handles.ROISize;
        sub_rois_del=handles.sub_rois_del;
        sub_rois=handles.sub_rois;
        img=handles.img;


      if ~isequal(size(sub_rois_del),size(sub_rois))
        h=msgbox({'Sub-ROI slection error in function subroi_avg_Callback'},...
          'Instruction','help','modal');
        waitfor(h)
      else
         Size = handles.ROISize.*handles.ROISize;
         sub_rois_del=handles.sub_rois_del;


        num_images = handles.number_images;
        c=handles.sub_rois_axis;

    for j=1:num_images

      if handles.mean == 2  % Calculates "No Mean method used"


        c=sub_rois{1};
        fft2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1, num_images);
        NPS2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1);
        count=0;

        for i=1:size(sub_rois,2)
          if ~sub_rois_del(i)
            c=sub_rois{i};

fft2d(:,:,j)=fft2d(:,:,j)+(abs(power(fftshift(fft2(double(img(c(3):c(4),c(1):c(2),j)))),2)))*((x_del*y_del)/Size
);
            count=count+1;
          end
```

```
    end
    fft2d(:,:,j)=fft2d(:,:,j)./count;
    NPS2d=NPS2d+fft2d(:,:,j);
    NPS2d = NPS2d./num_images;
  end




  if handles.mean == 0  % Calculates "grand mean"

    roi_mean=mean(mean(img(c(3):c(4),c(1):c(2),j)));
    img(c(3):c(4),c(1):c(2),j)=img(c(3):c(4),c(1):c(2),j)-roi_mean;

    c=sub_rois{1};
    fft2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1, num_images);
    NPS2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1);
    count=0;

    for i=1:size(sub_rois,2)
      if ~sub_rois_del(i)
        c=sub_rois{i};

fft2d(:,:,j)=fft2d(:,:,j)+(abs(power(fftshift(fft2(double(img(c(3):c(4),c(1):c(2),j)))),2)))*((x_del*y_del)/Size
);
        count=count+1;
      end
    end
    fft2d(:,:,j)=fft2d(:,:,j)./count;
    NPS2d=NPS2d+fft2d(:,:,j);
    NPS2d = NPS2d./num_images;
  end

  if handles.mean == 1  % Calculates "local mean"



    c=sub_rois{1};
    fft2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1, num_images);
    NPS2d=zeros(c(4)-c(3)+1,c(2)-c(1)+1);
    count=0;

    for i=1:size(sub_rois,2)
      if ~sub_rois_del(i)
        c=sub_rois{i};
        roi_mean=mean(mean(img(c(3):c(4),c(1):c(2),j)));
        img(c(3):c(4),c(1):c(2),j)=img(c(3):c(4),c(1):c(2),j)-roi_mean;

fft2d(:,:,j)=fft2d(:,:,j)+(abs(power(fftshift(fft2(double(img(c(3):c(4),c(1):c(2),j)))),2)))*((x_del*y_del)/Size
);
        count=count+1;
      end
    end
    fft2d(:,:,j)=fft2d(:,:,j)./count;
    NPS2d=NPS2d+fft2d(:,:,j);
```

```
    NPS2d = NPS2d./num_images;
   end



  end

   end

   handles.sub_rois_fft2d=NPS2d;%sub_rois_fft2d;
   guidata(hObject,handles);

end


   %----------
   %sub_rois_avg=handles.sub_rois_avg;
   %sub_rois_fft2d=abs(fftshift(fft2(sub_rois_avg)));

  %  handles.sub_rois_fft2d=fft2d;%sub_rois_fft2d;
  %  guidata(hObject,handles);

   set(handles.plot,'Value',...
     strmatch({'2D NPS'},get(handles.plot,'String'),'exact'))
   %       strmatch({'ROI image'},get(plot_obj,'String'),'exact'))
   plot_Callback(handles.plot,eventdata, handles)

%--------------------------------------------------------------------------

% --- Executes on button press in nps_plots.
function nps_plots_Callback(hObject, eventdata, handles)
    if ~isfield(handles,'img')  || ~isfield(handles,'sub_rois_fft2d')
      h=msgbox({'The FFT has not yet been calculated'},...
        'Instruction','help','modal');
      waitfor(h)
       return
    end


   fft2d=handles.sub_rois_fft2d;
   s=round((size(fft2d)+1)./2);


   prompt = {['Enter number of lines on either side of the center line to average.']};
   dlg_title = 'NPS';
   %num_lines=[1,30];
   num_lines=1;
   def = {'3'};
   options.Resize='on';
   options.WindowStyle='modal';
   options.Interpreter='tex';
   ans  = inputdlg(prompt,dlg_title,num_lines,def,options);
   nl=str2num(char(ans))';
```

```
if isempty(nl)
  return
elseif nl < 0 || nl > s(1) || nl > s(2)
  h=msgbox({'Invalid value, value of 3 is being used'},...
    'Instruction','help','modal');
  waitfor(h)
  nl=3;
end
nps_vt=mean(fft2d(:,[s(1)-nl:s(1)-1 s(1)+1:s(1)+nl])');
nps_hz=mean(fft2d([s(2)-nl:s(2)-1 s(2)+1:s(2)+nl],:));
handles.nps_hz=nps_hz;
handles.nps_vt=nps_vt;
handles.nps_num_lines=nl;
guidata(hObject,handles);

if ~isfield(handles,'ROISize')
  h=msgbox({'Currently not functioning properly if a non square image or ROI has not been slected'},...
    'Instruction','help','modal');
  waitfor(h)
%     return

  N=size(diag(fft2d),1);
  handles.ROISize=N;

 NPS_45 = [nl*2, N-1];

for i = 1:nl*2
   if i<nl+1
      NPS_45(i,i+1:N) = diag(fft2d,i);
   else
      k=abs(nl-i);
      NPS_45(i,k+1:N) = diag(fft2d,-k);
   end
end
 % averaging across the diag

nps_45 = mean(NPS_45,1);
handles.nps_45=nps_45;
guidata(hObject,handles);

  set(handles.plot,'Value',...
  strmatch({'NPS HZ+VT+45'},get(handles.plot,'String'),'exact'))
%       strmatch({'ROI image'},get(plot_obj,'String'),'exact'))
plot_Callback(handles.plot,eventdata, handles)

end

%Used when a square matrix (sub-ROI) has been selected
 if isfield(handles,'ROISize')

  N = handles.ROISize;
  n_45 = (N/2)-2;
```

```
 NPS_45 = [nl*2, N-1];

nps_45=fft2d([s(2)+1:N],[s(1)+1:N]);


for i = 1:nl*2
   if i<nl+1
      NPS_45(i,i:n_45) = diag(nps_45,i);
   else
      k=abs(nl-i);
      NPS_45(i,k:n_45) = diag(nps_45,-k);
   end

end
% averaging across the diag

%   nps_45_mean = mean(NPS_45,1);
for i=1:nl-1
   nps_45_mean(1,i)=(sum(NPS_45(:,i)))/(2*i);
end

nps_45_mean(1,nl:n_45) = mean(NPS_45(:,nl:n_45));



handles.nps_45=nps_45_mean;
guidata(hObject,handles);

  set(handles.plot,'Value',...
  strmatch({'NPS HZ+VT+45'},get(handles.plot,'String'),'exact'))
%       strmatch({'ROI image'},get(plot_obj,'String'),'exact'))
plot_Callback(handles.plot,eventdata, handles)


end




% --- Executes on selection change in Mean_menu.
function Mean_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu7 contents as cell array
%       contents{get(hObject,'Value')} returns selected item from popupmenu7



contents = get(hObject,'String');
```

```
switch contents{get(hObject,'Value')}
    case ('Grand Mean')
        handles.mean = 0;
    case ('Local Mean')
        handles.mean = 1;
    case ('No Correction')
        handles.mean = 2;

end

guidata(hObject,handles);


% --- Executes during object creation, after setting all properties.
function Mean_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1
```